

Management Information Base for Network Management
of TCP/IP-based internets:
MIB-II

1. Status of this Memo

This memo defines the second version of the Management Information Base (MIB-II) for use with network management protocols in TCP/IP-based internets. In particular, together with its companion memos which describe the structure of management information (RFC 1155) along with the network management protocol (RFC 1157) for TCP/IP-based internets, these documents provide a simple, workable architecture and system for managing TCP/IP-based internets and in particular the Internet community.

This document on MIB-II incorporates all of the technical content of RFC 1156 on MIB-I and extends it, without loss of compatibility. However, MIB-I as described in RFC 1156 is full Standard Protocol of the Internet, while the MIB-II described here is Proposed Standard Protocol of the Internet.

This memo defines a mandatory extension to the base MIB (RFC 1156) and is a Proposed Standard for the Internet community. The extensions described here are currently Elective, but when they become a standard, they will have the same status as RFC 1156, that is, Recommended. The Internet Activities Board recommends that all IP and TCP implementations be network manageable. This implies implementation of the Internet MIB (RFC 1156 and the extensions in RFC 1158) and at least one of the two recommended management protocols SNMP (RFC 1157) or CMOT (RFC 1095).

This version of the MIB specification, MIB-II, is an incremental refinement of MIB-I. As such, it has been designed according to two criteria: first, changes have been made in response to new operational requirements in the Internet; and, second, the changes are entirely upwards compatible in order to minimize impact on the network as the managed nodes in the Internet transition from MIB-I to MIB-II.

It is expected that additional MIB groups and variables will be defined over time to accommodate the monitoring and control needs of new or changing components of the Internet.

Please refer to the latest edition of the "IAB Official Protocol Standards" RFC for current information on the state and status of standard Internet protocols.

Distribution of this memo is unlimited.

Table of Contents

1. Status of this Memo	1
2. Introduction	3
3. Changes from MIB-I	4
3.1 Deprecated Objects	4
3.2 Display Strings	5
3.3 The System Group	5
3.4 The Interfaces Group	5
3.5 The Address Translation Group	6
3.6 The IP Group	7
3.7 The ICMP Group	7
3.8 The TCP Group	7
3.9 The UDP Group	7
3.10 The EGP Group	8
3.11 The Transmission Group	8
3.12 The SNMP Group	8
4. Objects	8
4.1 Object Groups	9
4.2 Format of Definitions	10
5. Object Definitions	10
5.1 The System Group	11
5.2 The Interfaces Group	14
5.2.1 The Interfaces table	15
5.3 The Address Translation Group	27
5.4 The IP Group	30
5.4.1 The IP Address table	38
5.4.2 The IP Routing table	41
5.4.3 The IP Address Translation table	48
5.5 The ICMP Group	51
5.6 The TCP Group	61
5.6.1 The TCP Connection table	66
5.6.2 Additional TCP Objects	69
5.7 The UDP Group	70
5.7.1 The UDP Listener table	72
5.8 The EGP Group	73
5.8.1 The EGP Neighbor table	75
5.8.2 Additional EGP variables	83
5.9 The Transmission Group	83
5.10 The SNMP Group	83
6. Definitions	95

7. Identification of OBJECT instances for use with the
SNMP 126

7.1 ifTable Object Type Names 127

7.2 atTable Object Type Names 127

7.3 ipAddrTable Object Type Names 128

7.4 ipRoutingTable Object Type Names 128

7.5 ipNetToMediaTable Object Type Names 129

7.6 tcpConnTable Object Type Names 129

7.7 udpTable Object Type Names 130

7.8 egpNeighTable Object Type Names 130

8. Acknowledgements 130

9. References 131

10. Security Considerations..... 133

11. Author's Address..... 133

2. Introduction

As reported in RFC 1052, IAB Recommendations for the Development of Internet Network Management Standards [1], a two-prong strategy for network management of TCP/IP-based internets was undertaken. In the short-term, the Simple Network Management Protocol (SNMP) was to be used to manage nodes in the Internet community. In the long-term, the use of the OSI network management framework was to be examined. Two documents were produced to define the management information: RFC 1065, which defined the Structure of Management Information (SMI) [2], and RFC 1066, which defined the Management Information Base (MIB) [3]. Both of these documents were designed so as to be compatible with both the SNMP and the OSI network management framework.

This strategy was quite successful in the short-term: Internet-based network management technology was fielded, by both the research and commercial communities, within a few months. As a result of this, portions of the Internet community became network manageable in a timely fashion.

As reported in RFC 1109, Report of the Second Ad Hoc Network Management Review Group [4], the requirements of the SNMP and the OSI network management frameworks were more different than anticipated. As such, the requirement for compatibility between the SMI/MIB and both frameworks was suspended. This action permitted the operational network management framework, the SNMP, to respond to new operational needs in the Internet community by producing this document.

As such, the current network management framework for TCP/IP-based internets consists of: Structure and Identification of

Management Information for TCP/IP-based internets, RFC 1155 [13], which describes how managed objects contained in the MIB are defined; Management Information Base for Network Management of TCP/IP-based internets (version 2), this memo, which describes the managed objects contained in the MIB; and, the Simple Network Management Protocol, RFC 1157 [14], which defines the protocol used to manage these objects.

Consistent with the IAB directive to produce simple, workable systems in the short-term, the list of objects (e.g., for BSD UNIX) were excluded.

- 7) It was agreed to avoid heavily instrumenting critical sections of code. The general guideline was one counter per critical section per layer.

3. Changes from MIB-I

Features of this MIB include:

- 1) incremental additions to reflect new operational requirements;
- 2) upwards compatibility with the SMI/MIB and the SNMP;
- 3) improved support for multi-protocol entities; and,
- 4) textual clean-up of the MIB to improve clarity and readability.

The objects defined in MIB-II have the OBJECT IDENTIFIER prefix:

```
mib-2      OBJECT IDENTIFIER ::= { mgmt 1 }
```

3.1. Deprecated Objects

In order to better prepare implementors for future changes in the MIB, a new term "deprecated" may be used when describing an object. A deprecated object in the MIB is one which must be supported, but one which will most likely be removed from the next version of the MIB (e.g., MIB-III).

MIB-II marks one object as being deprecated:

atTable

As a result of deprecating the atTable object, the entire Address Translation group is deprecated.

Note that no functionality is lost with the deprecation of these objects: new objects providing equivalent or superior functionality are defined in MIB-II.

3.2. Display Strings

In the past, there have been misinterpretations of the MIB as to when a string of octets should contain printable characters, meant to be displayed to a human. As a textual convention in the MIB, the datatype

```
DisplayString ::= OCTET STRING
```

is introduced. A DisplayString is restricted to the NVT ASCII character set, as defined in pages 10-11 of [7].

The following objects are now defined in terms of DisplayString:

```
sysDescr  
ifDescr
```

It should be noted that this change has no effect on either the syntax nor semantics of these objects. The use of the DisplayString notation is merely an artifact of the explanatory method used in MIB-II and future MIBs.

Further, it should be noted that any object defined in terms of OCTET STRING may contain arbitrary binary data, in which each octet may take any value from 0 to 255 (decimal).

3.3. The System Group

Four new objects are added to this group:

```
sysContact  
sysName  
sysLocation  
sysServices
```

These provide contact, administrative, location, and service information regarding the managed node.

3.4. The Interfaces Group

The definition of the ifNumber object was incorrect, as it required all interfaces to support IP. (For example, devices without IP, such as MAC-layer bridges, could not be managed if this definition was strictly followed.) The description of the ifNumber object is changed

accordingly.

The ifTable object was mistakenly marked as read-write, it has been (correctly) re-designated as read-only. In addition, several new values have been added to the ifType column in the ifTable object:

```
ppp(23)
softwareLoopback(24)
eon(25)
ethernet-3Mbit(26)
nsip(27)
slip(28)
```

Finally, a new column has been added to the ifTable object:

```
ifSpecific
```

which provides information about information specific to the media being used to realize the interface.

3.5. The Address Translation Group

In MIB-I, this group contained a table which permitted mappings from network addresses (e.g., IP addresses) to physical addresses (e.g., MAC addresses). Experience has shown that efficient implementations of this table make two assumptions: a single network protocol environment, and mappings occur only from network address to physical address.

The need to support multi-protocol nodes (e.g., those with both the IP and CLNP active), and the need to support the inverse mapping (e.g., for ES-IS), have invalidated both of these assumptions. As such, the atTable object is declared deprecated.

In order to meet both the multi-protocol and inverse mapping requirements, MIB-II and its successors will allocate up to two address translation tables inside each network protocol group. That is, the IP group will contain one address translation table, for going from IP addresses to physical addresses. Similarly, when a document defining MIB objects for the CLNP is produced (e.g., [8]), it will contain two tables, for mappings in both directions, as this is required for full functionality.

It should be noted that the choice of two tables (one for each direction of mapping) provides for ease of implementation in many cases, and does not introduce undue burden on implementations which realize the address translation abstraction through a single internal table.

3.6. The IP Group

The access attribute of the variable ipForwarding has been changed from read-only to read-write.

In addition, there is a new column to the ipAddrTable object,

ipAdEntReasmMaxSize

which keeps track of the largest IP datagram that can be re-assembled on a particular interface. There is also a new column in the ipRoutingTable object,

ipRouteMask

which is used for IP routing subsystems that support arbitrary subnet masks.

One new object is added to the IP group:

ipNetToMediaTable

which is the address translation table for the IP group (providing identical functionality to the now deprecated atTable in the address translation group).

3.7. The ICMP Group

There are no changes to this group.

3.8. The TCP Group

Two new variables are added:

tcpInErrs
tcpOutRsts

which keep track of the number of incoming TCP segments in error and the number of resets generated by a TCP.

3.9. The UDP Group

A new table:

udpTable

is added.

3.10. The EGP Group

Experience has indicated a need for additional objects that are useful in EGP monitoring. In addition to making several additions to the `egpNeighborTable` object, a new variable is added:

```
egpAs
```

which gives the autonomous system associated with this EGP entity.

3.11. The Transmission Group

MIB-I was lacking in that it did not distinguish between different types of transmission media. A new group, the Transmission group, is allocated for this purpose:

```
transmission OBJECT IDENTIFIER ::= { mib-2 10 }
```

When Internet-standard definitions for managing transmission media are defined, the transmission group is used to provide a prefix for the names of those objects.

Typically, such definitions reside in the experimental portion of the MIB until they are "proven", then as a part of the Internet standardization process, the definitions are accordingly elevated and a new object identifier, under the transmission group is defined. By convention, the name assigned is:

```
type OBJECT IDENTIFIER ::= { transmission number }
```

where "type" is the symbolic value used for the media in the `ifType` column of the `ifTable` object, and "number" is the actual integer value corresponding to the symbol.

3.12. The SNMP Group

The application-oriented working groups of the IETF have been tasked to be receptive towards defining MIB variables specific to their respective applications.

For the SNMP, it is useful to have statistical information. A new group, the SNMP group, is allocated for this purpose:

```
snmp OBJECT IDENTIFIER ::= { mib-2 11 }
```

4. Objects

Managed objects are accessed via a virtual information store, termed

the Management Information Base or MIB. Objects in the MIB are defined using Abstract Syntax Notation One (ASN.1) [9].

The mechanisms used for describing these objects are specified the companion memo, the SMI. In particular, each object has a name, a syntax, and an encoding. The name is an object identifier, an administratively assigned name, which specifies an object type. 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 OBJECT DESCRIPTOR, to also refer to the object type.

The syntax of an object type defines the abstract data structure corresponding to that object type. The ASN.1 language is used for this purpose. However, the companion memo purposely restricts the ASN.1 constructs which may be used. These restrictions are explicitly made for simplicity.

The encoding of an object type is simply how that object type is represented using the object type's syntax. Implicitly tied to the notion of an object type's syntax and encoding is how the object type is represented when being transmitted on the network. This memo specifies the use of the basic encoding rules (BER) of ASN.1 [10], subject to the additional requirements imposed by the SNMP [14].

4.1. Object Groups

Since this list of managed objects contains only the essential elements, there is no need to allow individual objects to be optional. Rather, the objects are arranged into the following groups:

- System
- Interfaces
- Address Translation (deprecated)
- IP
- ICMP
- TCP
- UDP
- EGP
- Transmission
- SNMP

There are two reasons for defining these groups: to provide a means of assigning object identifiers; and, to provide a method for implementations of managed agents to know which objects they must implement. This method is as follows: if the semantics of a group is applicable to an implementation, then it must implement all objects

in that group. For example, an implementation must implement the EGP group if and only if it implements the EGP.

4.2. Format of Definitions

The next section contains the specification of all object types contained in the MIB. Following the conventions of the companion memo, the object types are defined using the following fields:

OBJECT:

A textual name, termed the OBJECT DESCRIPTOR, for the object type, along with its corresponding OBJECT IDENTIFIER.

Syntax:

The abstract syntax for the object type, presented using ASN.1. This must resolve to an instance of the ASN.1 type ObjectSyntax defined in the SMI.

Definition:

A textual description of the semantics of the object type. Implementations should ensure that their interpretation of the object type fulfills this definition since this MIB is intended for use in multi-vendor environments. As such it is vital that object types have consistent meaning across all machines.

Access:

A keyword, one of read-only, read-write, write-only, or not-accessible. Note that this designation specifies the minimum level of support required. As a local matter, implementations may support other access types (e.g., an implementation may elect to permitting writing a variable marked herein as read-only). Further, protocol-specific "views" (e.g., those implied by an SNMP community) may make further restrictions on access to a variable.

Status:

A keyword, one of mandatory, optional, obsolete, or deprecated. Use of deprecated implies mandatory status.

5. Object Definitions

RFC1158-MIB

DEFINITIONS ::= BEGIN

```

IMPORTS
    mgmt, OBJECT-TYPE, NetworkAddress, IpAddress,
    Counter, Gauge, TimeTicks
    FROM RFC1155-SMI;

DisplayString ::=
    OCTET STRING

mib-2      OBJECT IDENTIFIER ::= { mgmt 1 }      -- MIB-II

system     OBJECT IDENTIFIER ::= { mib-2 1 }
interfaces OBJECT IDENTIFIER ::= { mib-2 2 }
at         OBJECT IDENTIFIER ::= { mib-2 3 }
ip         OBJECT IDENTIFIER ::= { mib-2 4 }
icmp       OBJECT IDENTIFIER ::= { mib-2 5 }
tcp        OBJECT IDENTIFIER ::= { mib-2 6 }
udp        OBJECT IDENTIFIER ::= { mib-2 7 }
egp        OBJECT IDENTIFIER ::= { mib-2 8 }
-- cmot    OBJECT IDENTIFIER ::= { mib-2 9 }
transmission OBJECT IDENTIFIER ::= { mib-2 10 }
snmp       OBJECT IDENTIFIER ::= { mib-2 11 }
END

```

5.1. The System Group

Implementation of the System group is mandatory for all systems.

OBJECT:

```

-----
    sysDescr { system 1 }

```

Syntax:

```

    DisplayString (SIZE (0..255))

```

Definition:

A textual description of the entity. This value should include the full name and version identification of the system's hardware type, software operating-system, and networking software. It is mandatory that this only contain printable ASCII characters.

Access:

read-only.

Status:

mandatory.

OBJECT:

sysObjectID { system 2 }

Syntax:

OBJECT IDENTIFIER

Definition:

The vendor's authoritative identification of the network management subsystem contained in the entity. This value is allocated within the SMI enterprises subtree (1.3.6.1.4.1) and provides an easy and unambiguous means for determining "what kind of box" is being managed. For example, if vendor "Flintstones, Inc." was assigned the subtree 1.3.6.1.4.1.4242, it could assign the identifier 1.3.6.1.4.1.4242.1.1 to its "Fred Router".

Access:

read-only.

Status:

mandatory.

OBJECT:

sysUpTime { system 3 }

Syntax:

TimeTicks

Definition:

The time (in hundredths of a second) since the network management portion of the system was last re-initialized.

Access:

read-only.

Status:

mandatory.

OBJECT:

sysContact { system 4 }

Syntax:

DisplayString (SIZE (0..255))

Definition:

The textual identification of the contact person for this managed node, together with information on how to contact this person.

Access:

read-write.

Status:

mandatory.

OBJECT:

sysName { system 5 }

Syntax:

DisplayString (SIZE (0..255))

Definition:

An administratively-assigned name for this managed node. By convention, this is the node's fully-qualified domain name.

Access:

read-write.

Status:

mandatory.

OBJECT:

sysLocation { system 6 }

Syntax:

DisplayString (SIZE (0..255))

Definition:

The physical location of this node (e.g., "telephone closet, 3rd floor").

Access:

read-only.

Status:

mandatory.

OBJECT:

sysServices { system 7 }

Syntax:

INTEGER (0..127)

Definition:

A value which indicates the set of services that this entity potentially offers. The value is a sum. This sum initially takes the value zero, Then, for each layer, L, in the range 1 through 7, that this node performs transactions for, 2^{L-1} is added to the sum. For example, a node which performs only routing functions would have a value of 4 (2^{3-1}). In contrast, a node which is a host offering application services would have a value of 72 ($2^{4-1} + 2^{7-1}$). Note that in the context of the Internet suite of protocols, values should be calculated accordingly:

layer	functionality
1	physical (e.g., repeaters)
2	datalink/subnetwork (e.g., bridges)
3	internet (e.g., supports the IP)
4	end-to-end (e.g., supports the TCP)
7	applications (e.g., supports the SMTP)

For systems including OSI protocols, layers 5 and 6 may also be counted.

Access:

read-only.

Status:

mandatory.

5.2. The Interfaces Group

Implementation of the Interfaces group is mandatory for all systems.

OBJECT:

ifNumber { interfaces 1 }

Syntax:

INTEGER

Definition:

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

Access:

read-only.

Status:

mandatory.

5.2.1. The Interfaces table

The Interfaces table contains information on the entity's interfaces. Each interface is thought of as being attached to a "subnetwork". Note that this term should not be confused with "subnet" which refers to an addressing partitioning scheme used in the Internet suite of protocols.

OBJECT:

ifTable { interfaces 2 }

Syntax:

SEQUENCE OF IfEntry

Definition:

A list of interface entries. The number of entries is given by the value of ifNumber.

Access:

read-only.

Status:

mandatory.

OBJECT:

ifEntry { ifTable 1 }

Syntax:

```
IfEntry ::= SEQUENCE {
    ifIndex
        INTEGER,
    ifDescr
        DisplayString,
    ifType
        INTEGER,
    ifMtu
        INTEGER,
    ifSpeed
        Gauge,
    ifPhysAddress
        OCTET STRING,
    ifAdminStatus
        INTEGER,
    ifOperStatus
        INTEGER,
    ifLastChange
        TimeTicks,
    ifInOctets
        Counter,
    ifInUcastPkts
        Counter,
    ifInNUcastPkts
        Counter,
    ifInDiscards
        Counter,
    ifInErrors
        Counter,
    ifInUnknownProtos
        Counter,
    ifOutOctets
        Counter,
    ifOutUcastPkts
        Counter,
    ifOutNUcastPkts
        Counter,
    ifOutDiscards
        Counter,
    ifOutErrors
        Counter,
    ifOutQLen
        Gauge,
    ifSpecific
        OBJECT IDENTIFIER
}
```


Definition:

An interface entry containing objects at the subnetwork layer and below for a particular interface.

Access:

read-only.

Status:

mandatory.

We now consider the individual components of each interface entry:

OBJECT:

ifIndex { ifEntry 1 }

Syntax:

INTEGER

Definition:

A unique value for each interface. 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.

Access:

read-only.

Status:

mandatory.

OBJECT:

ifDescr { ifEntry 2 }

Syntax:

DisplayString (SIZE (0..255))

Definition:

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 hardware interface.

Access:
read-only.

Status:
mandatory.

OBJECT:

ifType { ifEntry 3 }

Syntax:

```

INTEGER {
    other(1),           -- none of the following
    regular1822(2),
    hdh1822(3),
    ddn-x25(4),
    rfc877-x25(5),
    ethernet-csmacd(6),
    iso88023-csmacd(7),
    iso88024-tokenBus(8),
    iso88025-tokenRing(9),
    iso88026-man(10),
    starLan(11),
    proteon-10Mbit(12),
    proteon-80Mbit(13),
    hyperchannel(14),
    fddi(15),
    lapb(16),
    sdlc(17),
    t1-carrier(18),
    cept(19),          -- european equivalent of T-1
    basicISDN(20),
    primaryISDN(21),
                                -- proprietary serial
    propPointToPointSerial(22),
    ppp(23),
    softwareLoopback(24),
    eon(25),           -- CLNP over IP [12]
    ethernet-3Mbit(26)
    nsip(27),          -- XNS over IP
    slip(28)           -- generic SLIP
}

```

Definition:

The type of interface, distinguished according to the physical/link protocol(s) immediately "below" the network layer in the protocol stack.

Access:
 read-only.

Status:
 mandatory.

OBJECT:

 ifMtu { ifEntry 4 }

Syntax:
 INTEGER

Definition:
 The size of the largest datagram 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.

Access:
 read-only.

Status:
 mandatory.

OBJECT:

 ifSpeed { ifEntry 5 }

Syntax:
 Gauge

Definition:
 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.

Access:
 read-only.

Status:
 mandatory.

OBJECT:

ifPhysAddress { ifEntry 6 }

Syntax:

OCTET STRING

Definition:

The interface's address at the protocol layer immediately "below" the network layer in the protocol stack. For interfaces which do not have such an address (e.g., a serial line), this object should contain an octet string of zero length.

Access:

read-only.

Status:

mandatory.

OBJECT:

ifAdminStatus { ifEntry 7 }

Syntax:

```
INTEGER {
    up(1),          -- ready to pass packets
    down(2),
    testing(3)     -- in some test mode
}
```

Definition:

The desired state of the interface. The testing(3) state indicates that no operational packets can be passed.

Access:

read-write.

Status:

mandatory.

OBJECT:

ifOperStatus { ifEntry 8 }

Syntax:

```
INTEGER {
    up(1),          -- ready to pass packets
    down(2),
    testing(3)     -- in some test mode
}
```

Definition:

The current operational state of the interface. The testing(3) state indicates that no operational packets can be passed.

Access:

read-only.

Status:

mandatory.

OBJECT:

```
ifLastChange { ifEntry 9 }
```

Syntax:

```
TimeTicks
```

Definition:

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.

Access:

read-only.

Status:

mandatory.

OBJECT:

```
ifInOctets { ifEntry 10 }
```

Syntax:

```
Counter
```

Definition:

The total number of octets received on the interface,
including framing characters.

Access:

read-only.

Status:

mandatory.

OBJECT:

ifInUcastPkts { ifEntry 11 }

Syntax:

Counter

Definition:

The number of subnetwork-unicast packets delivered to a
higher-layer protocol.

Access:

read-only.

Status:

mandatory.

OBJECT:

ifInNUcastPkts { ifEntry 12 }

Syntax:

Counter

Definition:

The number of non-unicast (i.e., subnetwork-broadcast or
subnetwork-multicast) packets delivered to a higher-layer
protocol.

Access:

read-only.

Status:

mandatory.

OBJECT:

ifInDiscards { ifEntry 13 }

Syntax:

Counter

Definition:

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.

Access:

read-only.

Status:

mandatory.

OBJECT:

ifInErrors { ifEntry 14 }

Syntax:

Counter

Definition:

The number of inbound packets that contained errors preventing them from being deliverable to a higher-layer protocol.

Access:

read-only.

Status:

mandatory.

OBJECT:

ifInUnknownProtos { ifEntry 15 }

Syntax:

Counter

Definition:

The number of packets received via the interface which were discarded because of an unknown or unsupported protocol.

Access:

read-only.

Status:

mandatory.

OBJECT:

ifOutOctets { ifEntry 16 }

Syntax:

Counter

Definition:

The total number of octets transmitted out of the interface, including framing characters.

Access:

read-only.

Status:

mandatory.

OBJECT:

ifOutUcastPkts { ifEntry 17 }

Syntax:

Counter

Definition:

The total number of packets that higher-level protocols requested be transmitted to a subnetwork-unicast address, including those that were discarded or not sent.

Access:

read-only.

Status:

mandatory.

OBJECT:

ifOutNUcastPkts { ifEntry 18 }

Syntax:

Counter

Definition:

The total number of packets that higher-level protocols requested be transmitted to a non-unicast (i.e., a subnetwork-broadcast or subnetwork-multicast) address, including those that were discarded or not sent.

Access:

read-only.

Status:

mandatory.

OBJECT:

ifOutDiscards { ifEntry 19 }

Syntax:

Counter

Definition:

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.

Access:

read-only.

Status:

mandatory.

OBJECT:

ifOutErrors { ifEntry 20 }

Syntax:

Counter

Definition:

The number of outbound packets that could not be transmitted because of errors.

Access:

read-only.

Status:

mandatory.

OBJECT:

ifOutQLen { ifEntry 21 }

Syntax:

Gauge

Definition:

The length of the output packet queue (in packets).

Access:

read-only.

Status:

mandatory.

OBJECT:

ifSpecific { ifEntry 22 }

Syntax:

OBJECT IDENTIFIER

Definition:

A reference to MIB definitions specific to the particular media being used to realize the interface. For example, if the interface is realized by an ethernet, then the value of this object refers to a document defining objects specific to ethernet. If an agent is not configured to have a value for any of these variables, the object identifier

 nullSpecific OBJECT IDENTIFIER ::= { 0 0 }

is returned. Note that "nullSpecific" is a syntatically valid object identifier, and any conformant

implementation of ASN.1 and BER must be able to generate and recognize this value.

Access:
read-only.

Status:
mandatory.

5.3. The Address Translation Group

Implementation of the Address Translation group is mandatory for all systems. Note however that this group is deprecated by MIB-II. That is, it is being included solely for compatibility with MIB-I nodes, and will most likely be excluded from MIB-III nodes. From MIB-II and onwards, each network protocol group contains its own address translation tables.

The Address Translation group contains one table which is the union across all interfaces of the translation tables for converting a NetworkAddress (e.g., an IP address) into a subnetwork-specific address. For lack of a better term, this document refers to such a subnetwork-specific address as a "physical" address.

Examples of such translation tables are: for broadcast media where ARP is in use, the translation table is equivalent to the ARP cache; or, on an X.25 network where non-algorithmic translation to X.121 addresses is required, the translation table contains the NetworkAddress to X.121 address equivalences.

OBJECT:

atTable { at 1 }

Syntax:
SEQUENCE OF AtEntry

Definition:
The Address Translation tables contain the NetworkAddress to "physical" address equivalences. Some interfaces do not use translation tables for determining address equivalences (e.g., DDN-X.25 has an algorithmic method); if all interfaces are of this type, then the Address Translation table is empty, i.e., has zero entries.

Access:
read-write.

Status:
depreciated.

OBJECT:

atEntry { atTable 1 }

Syntax:

```
AtEntry ::= SEQUENCE {
    atIfIndex
        INTEGER,
    atPhysAddress
        OCTET STRING,
    atNetAddress
        NetworkAddress
}
```

Definition:

Each entry contains one NetworkAddress to "physical" address equivalence.

Access:

read-write.

Status:

depreciated.

We now consider the individual components of each Address Translation table entry:

OBJECT:

atIfIndex { atEntry 1 }

Syntax:

```
INTEGER
```

Definition:

The interface on which this entry's equivalence is effective. The interface identified by a particular value of this index is the same interface as identified by the same value of ifIndex.

Access:

read-write.

Status:
depreciated.

OBJECT:

atPhysAddress { atEntry 2 }

Syntax:
OCTET STRING

Definition:
The media-dependent "physical" address.

Setting this object to a null string (one of zero length) has the effect of invaliding the corresponding entry in the atTable object. That is, it effectively disassociates the interface identified with said entry from the mapping identified with said entry. It is an implementation-specific matter as to whether the agent removes an invalidated entry from the table. Accordingly, management stations must be prepared to receive tabular information from agents that corresponds to entries not currently in use. Proper interpretation of such entries requires examination of the relevant atPhysAddress object.

Access:
read-write.

Status:
depreciated.

OBJECT:

atNetAddress { atEntry 3 }

Syntax:
NetworkAddress

Definition:
The NetworkAddress (e.g., the IP address) corresponding to the media-dependent "physical" address.

Access:
read-write.

Status:
depreciated.

5.4. The IP Group

Implementation of the IP group is mandatory for all systems.

OBJECT:

ipForwarding { ip 1 }

Syntax:

```
INTEGER {
    forwarding(1),    -- i.e., acting as a gateway
    not-forwarding(2) -- i.e., NOT acting as a gateway
}
```

Definition:

The indication of whether this entity is acting as an IP gateway in respect to the forwarding of datagrams received by, but not addressed to, this entity. IP gateways forward datagrams. IP hosts do not (except those source-routed via the host).

Access:

read-write.

Status:

mandatory.

OBJECT:

ipDefaultTTL { ip 2 }

Syntax:

```
INTEGER
```

Definition:

The default value inserted into the Time-To-Live field of the IP header of datagrams originated at this entity, whenever a TTL value is not supplied by the transport layer protocol.

Access:

read-write.

Status:
mandatory.

OBJECT:

ipInReceives { ip 3 }

Syntax:
Counter

Definition:
The total number of input datagrams received from
interfaces, including those received in error.

Access:
read-only.

Status:
mandatory.

OBJECT:

ipInHdrErrors { ip 4 }

Syntax:
Counter

Definition:
The number of input datagrams discarded due to errors in
their IP headers, including bad checksums, version number
mismatch, other format errors, time-to-live exceeded,
errors discovered in processing their IP options, etc.

Access:
read-only.

Status:
mandatory.

OBJECT:

ipInAddrErrors { ip 5 }

Syntax:
Counter

Definition:

The number of input datagrams discarded because the IP address in their IP header's destination field was not a valid address to be received at this entity. This count includes invalid addresses (e.g., 0.0.0.0) and addresses of unsupported Classes (e.g., Class E). For entities which are not IP Gateways and therefore do not forward datagrams, this counter includes datagrams discarded because the destination address was not a local address.

Access:

read-only.

Status:

mandatory.

OBJECT:

ipForwDatagrams { ip 6 }

Syntax:

Counter

Definition:

The number of input datagrams for which this entity was not their final IP destination, as a result of which an attempt was made to find a route to forward them to that final destination. In entities which do not act as IP Gateways, this counter will include only those packets which were Source-Routed via this entity, and the Source-Route option processing was successful.

Access:

read-only.

Status:

mandatory.

OBJECT:

ipInUnknownProtos { ip 7 }

Syntax:

Counter

Definition:

The number of locally-addressed datagrams received successfully but discarded because of an unknown or unsupported protocol.

Access:

read-only.

Status:

mandatory.

OBJECT:

ipInDiscards { ip 8 }

Syntax:

Counter

Definition:

The number of input IP datagrams for which no problems were encountered to prevent their continued processing, but which were discarded (e.g., for lack of buffer space). Note that this counter does not include any datagrams discarded while awaiting re-assembly.

Access:

read-only.

Status:

mandatory.

OBJECT:

ipInDelivers { ip 9 }

Syntax:

Counter

Definition:

The total number of input datagrams successfully delivered to IP user-protocols (including ICMP).

Access:

read-only.

Status:
mandatory.

OBJECT:

ipOutRequests { ip 10 }

Syntax:
Counter

Definition:

The total number of IP datagrams which local IP user-protocols (including ICMP) supplied to IP in requests for transmission. Note that this counter does not include any datagrams counted in ipForwDatagrams.

Access:
read-only.

Status:
mandatory.

OBJECT:

ipOutDiscards { ip 11 }

Syntax:
Counter

Definition:

The number of output IP datagrams for which no problem was encountered to prevent their transmission to their destination, but which were discarded (e.g., for lack of buffer space). Note that this counter would include datagrams counted in ipForwDatagrams if any such packets met this (discretionary) discard criterion.

Access:
read-only.

Status:
mandatory.

OBJECT:

ipOutNoRoutes { ip 12 }

Syntax:
Counter

Definition:
The number of IP datagrams discarded because no route could be found to transmit them to their destination. Note that this counter includes any packets counted in ipForwDatagrams which meet this "no-route" criterion. Note that this includes any datagrams which a host cannot route because all of its default gateways are down.

Access:
read-only.

Status:
mandatory.

OBJECT:

ipReasmTimeout { ip 13 }

Syntax:
INTEGER

Definition:
The maximum number of seconds which received fragments are held while they are awaiting reassembly at this entity.

Access:
read-only.

Status:
mandatory.

OBJECT:

ipReasmReqds { ip 14 }

Syntax:
Counter

Definition:
The number of IP fragments received which needed to be reassembled at this entity.

Access:
 read-only.

Status:
 mandatory.

OBJECT:

 ipReasmOKs { ip 15 }

Syntax:
 Counter

Definition:
 The number of IP datagrams successfully re-assembled.

Access:
 read-only.

Status:
 mandatory.

OBJECT:

 ipReasmFails { ip 16 }

Syntax:
 Counter

Definition:
 The number of failures detected by the IP re-assembly algorithm (for whatever reason: timed out, errors, etc). Note that this is not necessarily a count of discarded IP fragments since some algorithms (notably the algorithm in RFC 815) can lose track of the number of fragments by combining them as they are received.

Access:
 read-only.

Status:
 mandatory.

OBJECT:

ipFragOKs { ip 17 }

Syntax:

Counter

Definition:

The number of IP datagrams that have been successfully fragmented at this entity.

Access:

read-only.

Status:

mandatory.

OBJECT:

ipFragFails { ip 18 }

Syntax:

Counter

Definition:

The number of IP datagrams that have been discarded because they needed to be fragmented at this entity but could not be, e.g., because their "Don't Fragment" flag was set.

Access:

read-only.

Status:

mandatory.

OBJECT:

ipFragCreates { ip 19 }

Syntax:

Counter

Definition:

The number of IP datagram fragments that have been generated as a result of fragmentation at this entity.

Access:
read-only.

Status:
mandatory.

5.4.1. The IP Address table

The Ip Address table contains this entity's IP addressing information.

OBJECT:

ipAddrTable { ip 20 }

Syntax:

SEQUENCE OF IpAddrEntry

Definition:

The table of addressing information relevant to this entity's IP addresses.

Access:
read-only.

Status:
mandatory.

OBJECT:

ipAddrEntry { ipAddrTable 1 }

Syntax:

```
IpAddrEntry ::= SEQUENCE {
    ipAdEntAddr
        IpAddress,
    ipAdEntIfIndex
        INTEGER,
    ipAdEntNetMask
        IpAddress,
    ipAdEntBcastAddr
        INTEGER,
    ipAdEntReasmMaxSize
        INTEGER (0..65535)
}
```

Definition:

The addressing information for one of this entity's IP addresses.

Access:

read-only.

Status:

mandatory.

OBJECT:

ipAdEntAddr { ipAddrEntry 1 }

Syntax:

IpAddress

Definition:

The IP address to which this entry's addressing information pertains.

Access:

read-only.

Status:

mandatory.

OBJECT:

ipAdEntIfIndex { ipAddrEntry 2 }

Syntax:

INTEGER

Definition:

The index value which uniquely identifies the interface to which this entry is applicable. The interface identified by a particular value of this index is the same interface as identified by the same value of ifIndex.

Access:

read-only.

Status:

mandatory.

OBJECT:

ipAdEntNetMask { ipAddrEntry 3 }

Syntax:

IpAddress

Definition:

The subnet mask associated with the IP address of this entry. The value of the mask is an IP address with all the network bits set to 1 and all the hosts bits set to 0.

Access:

read-only.

Status:

mandatory.

OBJECT:

ipAdEntBcastAddr { ipAddrEntry 4 }

Syntax:

INTEGER

Definition:

The value of the least-significant bit in the IP broadcast address used for sending datagrams on the (logical) interface associated with the IP address of this entry. For example, when the Internet standard all-ones broadcast address is used, the value will be 1. This value applies to both the subnet and network broadcasts addresses used by the entity on this (logical) interface.

Access:

read-only.

Status:

mandatory.

OBJECT:

ipAdEntReasmMaxSize { ipAddrEntry 5 }

Syntax:

```
INTEGER (0..65535)
```

Definition:

The size of the largest IP datagram which this entity can re-assemble from incoming IP fragmented datagrams received on this interface.

Access:

```
read-only.
```

Status:

```
mandatory.
```

5.4.2. The IP Routing table

The IP Routing table contains an entry for each route presently known to this entity.

OBJECT:

```
-----
```

```
ipRoutingTable { ip 21 }
```

Syntax:

```
SEQUENCE OF IpRouteEntry
```

Definition:

This entity's IP Routing table.

Access:

```
read-write.
```

Status:

```
mandatory.
```

OBJECT:

```
-----
```

```
ipRouteEntry { ipRoutingTable 1 }
```

Syntax:

```
IpRouteEntry ::= SEQUENCE {
    ipRouteDest
        IpAddress,
    ipRouteIfIndex
        INTEGER,
    ipRouteMetric1
```

```
        INTEGER,  
        ipRouteMetric2  
        INTEGER,  
        ipRouteMetric3  
        INTEGER,  
        ipRouteMetric4  
        INTEGER,  
        ipRouteNextHop  
        IpAddress,  
        ipRouteType  
        INTEGER,  
        ipRouteProto  
        INTEGER,  
        ipRouteAge  
        INTEGER,  
        ipRouteMask  
        IpAddress  
    }
```

Definition:

A route to a particular destination.

Access:

read-write.

Status:

mandatory.

We now consider the individual components of each route in the IP Routing table:

OBJECT:

```
        ipRouteDest { ipRouteEntry 1 }
```

Syntax:

```
        IpAddress
```

Definition:

The destination IP address of this route. An entry with a value of 0.0.0.0 is considered a default route. Multiple routes to a single destination can appear in the table, but access to such multiple entries is dependent on the table-access mechanisms defined by the network management protocol in use.

Access:
 read-write.

Status:
 mandatory.

OBJECT:

 ipRouteIfIndex { ipRouteEntry 2 }

Syntax:
 INTEGER

Definition:
 The index value which uniquely identifies the local interface through which the next hop of this route should be reached. The interface identified by a particular value of this index is the same interface as identified by the same value of ifIndex.

Access:
 read-write.

Status:
 mandatory.

OBJECT:

 ipRouteMetric1 { ipRouteEntry 3 }

Syntax:
 INTEGER

Definition:
 The primary routing metric for this route. The semantics of this metric are determined by the routing-protocol specified in the route's ipRouteProto value. If this metric is not used, its value should be set to -1.

Access:
 read-write.

Status:
 mandatory.

OBJECT:

ipRouteMetric2 { ipRouteEntry 4 }

Syntax:

INTEGER

Definition:

An alternate routing metric for this route. The semantics of this metric are determined by the routing-protocol specified in the route's ipRouteProto value. If this metric is not used, its value should be set to -1.

Access:

read-write.

Status:

mandatory.

OBJECT:

ipRouteMetric3 { ipRouteEntry 5 }

Syntax:

INTEGER

Definition:

An alternate routing metric for this route. The semantics of this metric are determined by the routing-protocol specified in the route's ipRouteProto value. If this metric is not used, its value should be set to -1.

Access:

read-write.

Status:

mandatory.

OBJECT:

ipRouteMetric4 { ipRouteEntry 6 }

Syntax:

INTEGER

Definition:

An alternate routing metric for this route. The semantics of this metric are determined by the routing-protocol specified in the route's ipRouteProto value. If this metric is not used, its value should be set to -1.

Access:

read-write.

Status:

mandatory.

OBJECT:

ipRouteNextHop { ipRouteEntry 7 }

Syntax:

IpAddress

Definition:

The IP address of the next hop of this route. (In the case of a route bound to an interface which is realized via a broadcast media, the value of this field is the agent's IP address on that interface.)

Access:

read-write.

Status:

mandatory.

OBJECT:

ipRouteType { ipRouteEntry 8 }

Syntax:

```

INTEGER {
    other(1),          -- none of the following
    invalid(2),       -- an invalidated route
    direct(3),        -- route to directly
                     -- connected (sub-)network
    remote(4)         -- route to a non-local
                     -- host/network/sub-network

```

```
    }
```

Definition:

The type of route.

Setting this object to the value invalid(2) has the effect of invalidating the corresponding entry in the ipRoutingTable object. That is, it effectively disassociates the destination identified with said entry from the route identified with said entry. It is an implementation-specific matter as to whether the agent removes an invalidated entry from the table. Accordingly, management stations must be prepared to receive tabular information from agents that corresponds to entries not currently in use. Proper interpretation of such entries requires examination of the relevant ipRouteType object.

Access:

read-write.

Status:

mandatory.

OBJECT:

```
    ipRouteProto { ipRouteEntry 9 }
```

Syntax:

```
    INTEGER {
        other(1),          -- none of the following
                           -- non-protocol information,
                           -- e.g., manually configured
        local(2),         -- entries
                           -- set via a network management
        netmgmt(3),       -- protocol
                           -- obtained via ICMP,
        icmp(4),          -- e.g., Redirect
                           -- the remaining values are
                           -- all gateway routing protocols
        egp(5),
        ggp(6),
        hello(7),
        rip(8),
        is-is(9),
```

```
    es-is(10),
    ciscoIgrp(11),
    bbnSpfIgp(12),
    ospf(13),
    bgp(14)
}
```

Definition:

The routing mechanism via which this route was learned. Inclusion of values for gateway routing protocols is not intended to imply that hosts should support those protocols.

Access:

read-only.

Status:

mandatory.

OBJECT:

```
ipRouteAge { ipRouteEntry 10 }
```

Syntax:

INTEGER

Definition:

The number of seconds since this route was last updated or otherwise determined to be correct. Note that no semantics of "too old" can be implied except through knowledge of the routing protocol by which the route was learned.

Access:

read-write.

Status:

mandatory.

OBJECT:

```
ipRouteMask { ipRouteEntry 11 }
```

Syntax:

IpAddress

Definition:

Indicate the mask to be logical-ANDed with the destination address before being compared to the value in the ipRouteDest field. For those systems that do not support arbitrary subnet masks, an agent constructs the value of the ipRouteMask by determining whether the value of the correspondent ipRouteDest field belong to a class-A, B, or C network, and then using one of:

mask	network
255.0.0.0	class-A
255.255.0.0	class-B
255.255.255.0	class-C

If the value of the ipRouteDest is 0.0.0.0 (a default route), then the mask value is also 0.0.0.0. It should be noted that all IP routing subsystems implicitly use this mechanism.

Access:

read-write.

Status:

mandatory.

5.4.3. The IP Address Translation table

The Address Translation tables contain the IpAddress to "physical" address equivalences. Some interfaces do not use translation tables for determining address equivalences (e.g., DDN-X.25 has an algorithmic method); if all interfaces are of this type, then the Address Translation table is empty, i.e., has zero entries.

OBJECT:

ipNetToMediaTable { ip 22 }

Syntax:

SEQUENCE OF IpNetToMediaEntry

Definition:

The IP Address Translation table used for mapping from IP addresses to physical addresses.

Access:

read-write.

Status:
mandatory.

OBJECT:

ipNetToMediaEntry { ipNetToMediaTable 1 }

Syntax:

```
ipNetToMediaEntry ::= SEQUENCE {
    ipNetToMediaIfIndex
        INTEGER,
    ipNetToMediaPhysAddress
        OCTET STRING,
    ipNetToMediaNetAddress
        IpAddress,
    ipNetToMediaType
        INTEGER
}
```

Definition:

Each entry contains one IpAddress to "physical" address equivalence.

Access:

read-write.

Status:

mandatory.

We now consider the individual components of each IP Address Translation table entry:

OBJECT:

ipNetToMediaIfIndex { ipNetToMediaEntry 1 }

Syntax:

```
INTEGER
```

Definition:

The interface on which this entry's equivalence is effective. The interface identified by a particular value of this index is the same interface as identified by the same value of ifIndex.

Access:
 read-write.

Status:
 mandatory.

OBJECT:

 ipNetToMediaPhysAddress { ipNetToMediaEntry 2 }

Syntax:
 OCTET STRING

Definition:
 The media-dependent "physical" address.

Access:
 read-write.

Status:
 mandatory.

OBJECT:

 ipNetToMediaNetAddress { ipNetToMediaEntry 3 }

Syntax:
 IpAddress

Definition:
 The IpAddress corresponding to the media-dependent
 "physical" address.

Access:
 read-write.

Status:
 mandatory.

OBJECT:

 ipNetToMediaType { ipNetToMediaEntry 4 }

Syntax:
 INTEGER {

```
        other(1),          -- none of the following
        invalid(2),       -- an invalidated mapping
        dynamic(3),
        static(4)
    }
```

Definition:

The type of mapping.

Setting this object to the value invalid(2) has the effect of invalidating the corresponding entry in the ipNetToMediaTable. That is, it effectively disassociates the interface identified with said entry from the mapping identified with said entry. It is an implementation-specific matter as to whether the agent removes an invalidated entry from the table. Accordingly, management stations must be prepared to receive tabular information from agents that corresponds to entries not currently in use. Proper interpretation of such entries requires examination of the relevant ipNetToMediaType object.

Access:

read-write.

Status:

mandatory.

5.5. The ICMP Group

Implementation of the ICMP group is mandatory for all systems.

The ICMP group contains the ICMP input and output statistics.

OBJECT:

```
    icmpInMsgs { icmp 1 }
```

Syntax:

Counter

Definition:

The total number of ICMP messages which the entity received. Note that this counter includes all those counted by icmpInErrors.

Access:
 read-only.

Status:
 mandatory.

OBJECT:

 icmpInErrors { icmp 2 }

Syntax:
 Counter

Definition:
 The number of ICMP messages which the entity received but determined as having ICMP-specific errors (bad ICMP checksums, bad length, etc.).

Access:
 read-only.

Status:
 mandatory.

OBJECT:

 icmpInDestUnreachs { icmp 3 }

Syntax:
 Counter

Definition:
 The number of ICMP Destination Unreachable messages received.

Access:
 read-only.

Status:
 mandatory.

OBJECT:

 icmpInTimeExcds { icmp 4 }

Syntax:
Counter

Definition:
The number of ICMP Time Exceeded messages received.

Access:
read-only.

Status:
mandatory.

OBJECT:

icmpInParmProbs { icmp 5 }

Syntax:
Counter

Definition:
The number of ICMP Parameter Problem messages received.

Access:
read-only.

Status:
mandatory.

OBJECT:

icmpInSrcQuenchs { icmp 6 }

Syntax:
Counter

Definition:
The number of ICMP Source Quench messages received.

Access:
read-only.

Status:
mandatory.

OBJECT:

icmpInRedirects { icmp 7 }

Syntax:

Counter

Definition:

The number of ICMP Redirect messages received.

Access:

read-only.

Status:

mandatory.

OBJECT:

icmpInEchos { icmp 8 }

Syntax:

Counter

Definition:

The number of ICMP Echo (request) messages received.

Access:

read-only.

Status:

mandatory.

OBJECT:

icmpInEchoReps { icmp 9 }

Syntax:

Counter

Definition:

The number of ICMP Echo Reply messages received.

Access:

read-only.

Status:
mandatory.

OBJECT:

icmpInTimestamps { icmp 10 }

Syntax:
Counter

Definition:
The number of ICMP Timestamp (request) messages received.

Access:
read-only.

Status:
mandatory.

OBJECT:

icmpInTimestampReps { icmp 11 }

Syntax:
Counter

Definition:
The number of ICMP Timestamp Reply messages received.

Access:
read-only.

Status:
mandatory.

OBJECT:

icmpInAddrMasks { icmp 12 }

Syntax:
Counter

Definition:
The number of ICMP Address Mask Request messages
received.

Access:
 read-only.

Status:
 mandatory.

OBJECT:

 icmpInAddrMaskReps { icmp 13 }

Syntax:
 Counter

Definition:
 The number of ICMP Address Mask Reply messages received.

Access:
 read-only.

Status:
 mandatory.

OBJECT:

 icmpOutMsgs { icmp 14 }

Syntax:
 Counter

Definition:
 The total number of ICMP messages which this entity
 attempted to send. Note that this counter includes all
 those counted by icmpOutErrors.

Access:
 read-only.

Status:
 mandatory.

OBJECT:

 icmpOutErrors { icmp 15 }

Syntax:
Counter

Definition:
The number of ICMP messages which this entity did not send due to problems discovered within ICMP such as a lack of buffers. This value should not include errors discovered outside the ICMP layer such as the inability of IP to route the resultant datagram. In some implementations there may be no types of error which contribute to this counter's value.

Access:
read-only.

Status:
mandatory.

OBJECT:

icmpOutDestUnreachs { icmp 16 }

Syntax:
Counter

Definition:
The number of ICMP Destination Unreachable messages sent.

Access:
read-only.

Status:
mandatory.

OBJECT:

icmpOutTimeExcds { icmp 17 }

Syntax:
Counter

Definition:
The number of ICMP Time Exceeded messages sent.

Access:
read-only.

Status:
mandatory.

OBJECT:

icmpOutParmProbs { icmp 18 }

Syntax:
Counter

Definition:
The number of ICMP Parameter Problem messages sent.

Access:
read-only.

Status:
mandatory.

OBJECT:

icmpOutSrcQuenchs { icmp 19 }

Syntax:
Counter

Definition:
The number of ICMP Source Quench messages sent.

Access:
read-only.

Status:
mandatory.

OBJECT:

icmpOutRedirects { icmp 20 }

Syntax:
Counter

Definition:
The number of ICMP Redirect messages sent. For a host,
this object will always be zero, since hosts do not send

redirects.

Access:
read-only.

Status:
mandatory.

OBJECT:

icmpOutEchos { icmp 21 }

Syntax:
Counter

Definition:
The number of ICMP Echo (request) messages sent.

Access:
read-only.

Status:
mandatory.

OBJECT:

icmpOutEchoReps { icmp 22 }

Syntax:
Counter

Definition:
The number of ICMP Echo Reply messages sent.

Access:
read-only.

Status:
mandatory.

OBJECT:

icmpOutTimestamps { icmp 23 }

Syntax:
Counter

Definition:
The number of ICMP Timestamp (request) messages sent.

Access:
read-only.

Status:
mandatory.

OBJECT:

icmpOutTimestampReps { icmp 24 }

Syntax:
Counter

Definition:
The number of ICMP Timestamp Reply messages sent.

Access:
read-only.

Status:
mandatory.

OBJECT:

icmpOutAddrMasks { icmp 25 }

Syntax:
Counter

Definition:
The number of ICMP Address Mask Request messages sent.

Access:
read-only.

Status:
mandatory.

OBJECT:

icmpOutAddrMaskReps { icmp 26 }

Syntax:

Counter

Definition:

The number of ICMP Address Mask Reply messages sent.

Access:

read-only.

Status:

mandatory.

5.6. The TCP Group

Implementation of the TCP group is mandatory for all systems that implement the TCP.

Note that instances of object types that represent information about a particular TCP connection are transient; they persist only as long as the connection in question.

OBJECT:

tcpRtoAlgorithm { tcp 1 }

Syntax:

```
INTEGER {
    other(1),      -- none of the following
    constant(2),  -- a constant rto
    rsre(3),      -- MIL-STD-1778, Appendix B
    vanj(4)       -- Van Jacobson's algorithm [11]
}
```

Definition:

The algorithm used to determine the timeout value used for retransmitting unacknowledged octets.

Access:

read-only.

Status:

mandatory.

OBJECT:

tcpRtoMin { tcp 2 }

Syntax:

INTEGER

Definition:

The minimum value permitted by a TCP implementation for the retransmission timeout, measured in milliseconds. More refined semantics for objects of this type depend upon the algorithm used to determine the retransmission timeout. In particular, when the timeout algorithm is rsre(3), an object of this type has the semantics of the LBOUND quantity described in RFC 793.

Access:

read-only.

Status:

mandatory.

OBJECT:

tcpRtoMax { tcp 3 }

Syntax:

INTEGER

Definition:

The maximum value permitted by a TCP implementation for the retransmission timeout, measured in milliseconds. More refined semantics for objects of this type depend upon the algorithm used to determine the retransmission timeout. In particular, when the timeout algorithm is rsre(3), an object of this type has the semantics of the UBOUND quantity described in RFC 793.

Access:

read-only.

Status:

mandatory.

OBJECT:

tcpMaxConn { tcp 4 }

Syntax:

INTEGER

Definition:

The limit on the total number of TCP connections the entity can support. In entities where the maximum number of connections is dynamic, this object should contain the value "-1".

Access:

read-only.

Status:

mandatory.

OBJECT:

tcpActiveOpens { tcp 5 }

Syntax:

Counter

Definition:

The number of times TCP connections have made a direct transition to the SYN-SENT state from the CLOSED state.

Access:

read-only.

Status:

mandatory.

OBJECT:

tcpPassiveOpens { tcp 6 }

Syntax:

Counter

Definition:

The number of times TCP connections have made a direct transition to the SYN-RCVD state from the LISTEN state.

Access:
 read-only.

Status:
 mandatory.

OBJECT:

 tcpAttemptFails { tcp 7 }

Syntax:
 Counter

Definition:
 The number of times TCP connections have made a direct transition to the CLOSED state from either the SYN-SENT state or the SYN-RCVD state, plus the number of times TCP connections have made a direct transition to the LISTEN state from the SYN-RCVD state.

Access:
 read-only.

Status:
 mandatory.

OBJECT:

 tcpEstabResets { tcp 8 }

Syntax:
 Counter

Definition:
 The number of times TCP connections have made a direct transition to the CLOSED state from either the ESTABLISHED state or the CLOSE-WAIT state.

Access:
 read-only.

Status:
 mandatory.

OBJECT:

tcpCurrEstab { tcp 9 }

Syntax:

Gauge

Definition:

The number of TCP connections for which the current state is either ESTABLISHED or CLOSE-WAIT.

Access:

read-only.

Status:

mandatory.

OBJECT:

tcpInSegs { tcp 10 }

Syntax:

Counter

Definition:

The total number of segments received, including those received in error. This count includes segments received on currently established connections.

Access:

read-only.

Status:

mandatory.

OBJECT:

tcpOutSegs { tcp 11 }

Syntax:

Counter

Definition:

The total number of segments sent, including those on current connections but excluding those containing only retransmitted octets.

Access:
 read-only.

Status:
 mandatory.

OBJECT:

 tcpRetransSegs { tcp 12 }

Syntax:
 Counter

Definition:
 The total number of segments retransmitted - that is, the number of TCP segments transmitted containing one or more previously transmitted octets.

Access:
 read-only.

Status:
 mandatory.

5.6.1. The TCP Connection table

The TCP connection table contains information about this entity's existing TCP connections.

OBJECT:

 tcpConnTable { tcp 13 }

Syntax:
 SEQUENCE OF TcpConnEntry

Definition:
 A table containing TCP connection-specific information.

Access:
 read-only.

Status:
 mandatory.

OBJECT:

tcpConnEntry { tcpConnTable 1 }

Syntax:

```
TcpConnEntry ::= SEQUENCE {
    tcpConnState
        INTEGER,
    tcpConnLocalAddress
        IpAddress,
    tcpConnLocalPort
        INTEGER (0..65535),
    tcpConnRemAddress
        IpAddress,
    tcpConnRemPort
        INTEGER (0..65535)
}
```

Definition:

Information about a particular current TCP connection. An object of this type is transient, in that it ceases to exist when (or soon after) the connection makes the transition to the CLOSED state.

Access:

read-only.

Status:

mandatory.

OBJECT:

tcpConnState { tcpConnEntry 1 }

Syntax:

```
INTEGER {
    closed(1),
    listen(2),
    synSent(3),
    synReceived(4),
    established(5),
    finWait1(6),
    finWait2(7),
    closeWait(8),
    lastAck(9),
    closing(10),
    timeWait(11)
}
```

}

Definition:

The state of this TCP connection.

Access:

read-only.

Status:

mandatory.

OBJECT:

tcpConnLocalAddress { tcpConnEntry 2 }

Syntax:

IpAddress

Definition:

The local IP address for this TCP connection. In the case of a connection in the listen state which is willing to accept connections for any IP interface associated with the node, the value 0.0.0.0 is used.

Access:

read-only.

Status:

mandatory.

OBJECT:

tcpConnLocalPort { tcpConnEntry 3 }

Syntax:

INTEGER (0..65535)

Definition:

The local port number for this TCP connection.

Access:

read-only.

Status:

mandatory.

OBJECT:

tcpConnRemAddress { tcpConnEntry 4 }

Syntax:

IpAddress

Definition:

The remote IP address for this TCP connection.

Access:

read-only.

Status:

mandatory.

OBJECT:

tcpConnRemPort { tcpConnEntry 5 }

Syntax:

INTEGER (0..65535)

Definition:

The remote port number for this TCP connection.

Access:

read-only.

Status:

mandatory.

5.6.2. Additional TCP Objects

OBJECT:

tcpInErrs { tcp 14 }

Syntax:

Counter

Definition:

The total number of segments received in error (e.g., bad TCP checksums).

Access:
 read-only.

Status:
 mandatory.

OBJECT:

 tcpOutRsts { tcp 15 }

Syntax:
 Counter

Definition:
 The number of TCP segments sent containing the RST flag.

Access:
 read-only.

Status:
 mandatory.

5.7. The UDP Group

Implementation of the UDP group is mandatory for all systems which implement the UDP.

OBJECT:

 udpInDatagrams { udp 1 }

Syntax:
 Counter

Definition:
 The total number of UDP datagrams delivered to UDP users.

Access:
 read-only.

Status:
 mandatory.

OBJECT:

udpNoPorts { udp 2 }

Syntax:

Counter

Definition:

The total number of received UDP datagrams for which there was no application at the destination port.

Access:

read-only.

Status:

mandatory.

OBJECT:

udpInErrors { udp 3 }

Syntax:

Counter

Definition:

The number of received UDP datagrams that could not be delivered for reasons other than the lack of an application at the destination port.

Access:

read-only.

Status:

mandatory.

OBJECT:

udpOutDatagrams { udp 4 }

Syntax:

Counter

Definition:

The total number of UDP datagrams sent from this entity.

Access:
 read-only.

Status:
 mandatory.

5.7.1. The UDP Listener table

The UDP listener table contains information about this entity's UDP end-points on which a local application is currently accepting datagrams.

OBJECT:

 udpTable { udp 5 }

Syntax:
 SEQUENCE OF UdpEntry

Definition:
 A table containing UDP listener information.

Access:
 read-only.

Status:
 mandatory.

OBJECT:

 udpEntry { udpTable 1 }

Syntax:
 UdpEntry ::= SEQUENCE {
 udpLocalAddress
 IpAddress,
 udpLocalPort
 INTEGER (0..65535)
 }

Definition:
 Information about a particular current UDP listener.

Access:
 read-only.

Status:
mandatory.

OBJECT:

udpLocalAddress { udpEntry 1 }

Syntax:
IpAddress

Definition:
The local IP address for this UDP listener. In the case of a UDP listener which is willing to accept datagrams for any IP interface associated with the node, the value 0.0.0.0 is used.

Access:
read-only.

Status:
mandatory.

OBJECT:

udpLocalPort { udpEntry 2 }

Syntax:
INTEGER (0..65535)

Definition:
The local port number for this UDP listener.

Access:
read-only.

Status:
mandatory.

5.8. The EGP Group

Implementation of the EGP group is mandatory for all systems which implement the EGP.

OBJECT:

egpInMsgs { egp 1 }

Syntax:

Counter

Definition:

The number of EGP messages received without error.

Access:

read-only.

Status:

mandatory.

OBJECT:

egpInErrors { egp 2 }

Syntax:

Counter

Definition:

The number of EGP messages received that proved to be in error.

Access:

read-only.

Status:

mandatory.

OBJECT:

egpOutMsgs { egp 3 }

Syntax:

Counter

Definition:

The total number of locally generated EGP messages.

Access:

read-only.

Status:
mandatory.

OBJECT:

egpOutErrors { egp 4 }

Syntax:
Counter

Definition:
The number of locally generated EGP messages not sent due to resource limitations within an EGP entity.

Access:
read-only.

Status:
mandatory.

5.8.1.1. The EGP Neighbor table

The Egp Neighbor table contains information about this entity's EGP neighbors.

OBJECT:

egpNeighTable { egp 5 }

Syntax:
SEQUENCE OF EgpNeighEntry

Definition:
The EGP neighbor table.

Access:
read-only.

Status:
mandatory.

OBJECT:

egpNeighEntry { egpNeighTable 1 }

Syntax:

```
EgpNeighEntry ::= SEQUENCE {
    egpNeighState
        INTEGER,
    egpNeighAddr
        IpAddress,
    egpNeighAs
        INTEGER,
    egpNeighInMsgs
        Counter,
    egpNeighInErrs
        Counter,
    egpNeighOutMsgs
        Counter,
    egpNeighOutErrs
        Counter,
    egpNeighInErrMsgs
        Counter,
    egpNeighOutErrMsgs
        Counter,
    egpNeighStateUps
        Counter,
    egpNeighStateDowns
        Counter,
    egpNeighIntervalHello
        INTEGER,
    egpNeighIntervalPoll
        INTEGER,
    egpNeighMode
        INTEGER,
    egpNeighEventTrigger
        INTEGER
}
```

Definition:

Information about this entity's relationship with a particular EGP neighbor.

Access:

read-only.

Status:

mandatory.

We now consider the individual components of each EGP neighbor entry:

OBJECT:

egpNeighState { egpNeighEntry 1 }

Syntax:

```
INTEGER {
    idle(1),
    acquisition(2),
    down(3),
    up(4),
    cease(5)
}
```

Definition:

The EGP state of the local system with respect to this entry's EGP neighbor. Each EGP state is represented by a value that is one greater than the numerical value associated with said state in RFC 904.

Access:

read-only.

Status:

mandatory.

OBJECT:

egpNeighAddr { egpNeighEntry 2 }

Syntax:

IpAddress

Definition:

The IP address of this entry's EGP neighbor.

Access:

read-only.

Status:

mandatory.

OBJECT:

egpNeighAs { egpNeighEntry 3 }

Syntax:
INTEGER

Definition:
The autonomous system of this EGP peer. Zero should be specified if the autonomous system number of the neighbor is not yet known.

Access:
read-only.

Status:
mandatory.

OBJECT:

egpNeighInMsgs { egpNeighEntry 4 }

Syntax:
Counter

Definition:
The number of EGP messages received without error from this EGP peer.

Access:
read-only.

Status:
mandatory.

OBJECT:

egpNeighInErrs { egpNeighEntry 5 }

Syntax:
Counter

Definition:
The number of EGP messages received from this EGP peer that proved to be in error (e.g., bad EGP checksum).

Access:
read-only.

Status:
mandatory.

OBJECT:

egpNeighOutMsgs { egpNeighEntry 6 }

Syntax:
Counter

Definition:
The number of locally generated EGP messages to this EGP peer.

Access:
read-only.

Status:
mandatory.

OBJECT:

egpNeighOutErrs { egpNeighEntry 7 }

Syntax:
Counter

Definition:
The number of locally generated EGP messages not sent to this EGP peer due to resource limitations within an EGP entity.

Access:
read-only.

Status:
mandatory.

OBJECT:

egpNeighInErrMsgs { egpNeighEntry 8 }

Syntax:
Counter

Definition:

The number of EGP-defined error messages received from this EGP peer.

Access:

read-only.

Status:

mandatory.

OBJECT:

egpNeighOutErrMsgs { egpNeighEntry 9 }

Syntax:

Counter

Definition:

The number of EGP-defined error messages sent to this EGP peer.

Access:

read-only.

Status:

mandatory.

OBJECT:

egpNeighStateUps { egpNeighEntry 10 }

Syntax:

Counter

Definition:

The number of EGP state transitions to the UP state with this EGP peer.

Access:

read-only.

Status:

mandatory.

OBJECT:

egpNeighStateDowns { egpNeighEntry 11 }

Syntax:

Counter

Definition:

The number of EGP state transitions from the UP state to any other state with this EGP peer.

Access:

read-only.

Status:

mandatory.

OBJECT:

egpNeighIntervalHello { egpNeighEntry 12 }

Syntax:

INTEGER

Definition:

The interval between EGP Hello command retransmissions (in hundredths of a second). This represents the t1 timer as defined in RFC 904.

Access:

read-only.

Status:

mandatory.

OBJECT:

egpNeighIntervalPoll { egpNeighEntry 13 }

Syntax:

INTEGER

Definition:

The interval between EGP poll command retransmissions (in hundredths of a second). This represents the t3 timer as defined in RFC 904.

Access:
read-only.

Status:
mandatory.

OBJECT:

egpNeighMode { egpNeighEntry 14 }

Syntax:

```
INTEGER {
    active(1),
    passive(2)
}
```

Definition:

The polling mode of this EGP entity, either passive or active.

Access:
read-only.

Status:
mandatory.

OBJECT:

egpNeighEventTrigger { egpNeighEntry 15 }

Syntax:

```
INTEGER {
    start(1),
    stop(2)
}
```

Definition:

A control variable used to trigger operator-initiated Start and Stop events. When read, this variable always returns the most recent value that egpNeighEventTrigger was set to. If it has not been set since the last initialization of the network management subsystem on the node, it returns a value of "stop".

Access:
read-write

Status:
mandatory.

5.8.2. Additional EGP variables

OBJECT:

egpAs { egp 6 }

Syntax:
INTEGER

Definition:
The autonomous system number of this EGP entity.

Access:
read-only.

Status:
mandatory.

5.9. The Transmission Group

Based on the transmission media underlying each interface on a system, the corresponding portion of the Transmission group is mandatory for that system.

When Internet-standard definitions for managing transmission media are defined, the transmission group is used to provide a prefix for the names of those objects.

Typically, such definitions reside in the experimental portion of the MIB until they are "proven", then as a part of the Internet standardization process, the definitions are accordingly elevated and a new object identifier, under the transmission group is defined. By convention, the name assigned is:

```
type OBJECT IDENTIFIER ::= { transmission number }
```

where "type" is the symbolic value used for the media in the ifType column of the ifTable object, and "number" is the actual integer value corresponding to the symbol.

5.10. The SNMP Group

Implementation of the SNMP group is mandatory for all systems which support an SNMP protocol entity. Some of the objects defined below

will be zero-valued in those SNMP implementations that are optimized to support only those functions specific to either a management agent or a management client.

OBJECT:

snmpInPkts { snmp 1 }

Syntax:

Counter

Definition:

The total number of PDUs delivered to the SNMP entity from the transport service.

Access:

read-only.

Status:

mandatory.

OBJECT:

snmpOutPkts { snmp 2 }

Syntax:

Counter

Definition:

The total number of SNMP PDUs which were passed from the SNMP protocol entity to the transport service.

Access:

read-only.

Status:

mandatory.

OBJECT:

snmpInBadVersions { snmp 3 }

Syntax:

Counter

Definition:

The total number of syntactically correct SNMP PDUs which were delivered to the SNMP protocol entity and were for an unsupported SNMP version.

Access:

read-only.

Status:

mandatory.

OBJECT:

snmpInBadCommunityNames { snmp 4 }

Syntax:

Counter

Definition:

The total number of SNMP PDUs delivered to the SNMP protocol entity which used a SNMP community name not known to said entity.

Access:

read-only.

Status:

mandatory.

OBJECT:

snmpInBadCommunityUses { snmp 5 }

Syntax:

Counter

Definition:

The total number of SNMP PDUs delivered to the SNMP protocol entity which represented an SNMP operation which was not allowed by the SNMP community named in the PDU.

Access:

read-only.

Status:

mandatory.

OBJECT:

snmpInASNParseErrs { snmp 6 }

Syntax:

Counter

Definition:

The total number of ASN.1 parsing errors (either in encoding or syntax) encountered by the SNMP protocol entity when decoding received SNMP PDUs.

Access:

read-only.

Status:

mandatory.

OBJECT:

snmpInBadTypes { snmp 7 }

Syntax:

Counter

Definition:

The total number of SNMP PDUs delivered to the SNMP protocol entity which had an unknown PDU type.

Access:

read-only.

Status:

mandatory.

OBJECT:

snmpInTooBigs { snmp 8 }

Syntax:

Counter

Definition:

The total number valid SNMP PDUs which were delivered to the SNMP protocol entity and for which the value of the "ErrorStatus" component is "tooBig."

Access:
 read-only.

Status:
 mandatory.

OBJECT:

 snmpInNoSuchNames { snmp 9 }

Syntax:
 Counter

Definition:
 The total number valid SNMP PDUs which were delivered to
 the SNMP protocol entity and for which the value of the
 "ErrorStatus" component is "noSuchName."

Access:
 read-only.

Status:
 mandatory.

OBJECT:

 snmpInBadValues { snmp 10 }

Syntax:
 Counter

Definition:
 The total number valid SNMP PDUs which were delivered to
 the SNMP protocol entity and for which the value of the
 "ErrorStatus" component is "badValue."

Access:
 read-only.

Status:
 mandatory.

OBJECT:

 snmpInReadOnlys { snmp 11 }

Syntax:
Counter

Definition:
The total number valid SNMP PDUs which were delivered to the SNMP protocol entity and for which the value of the "ErrorStatus" component is "readOnly."

Access:
read-only.

Status:
mandatory.

OBJECT:

snmpInGenErrs { snmp 12 }

Syntax:
Counter

Definition:
The total number valid SNMP PDUs which were delivered to the SNMP protocol entity and for which the value of the "ErrorStatus" component is "genErr."

Access:
read-only.

Status:
mandatory.

OBJECT:

snmpInTotalReqVars { snmp 13 }

Syntax:
Counter

Definition:
The total number of MIB objects which have been retrieved successfully by the SNMP protocol entity as the result of receiving valid SNMP Get-Request and Get-Next PDUs.

Access:
read-only.

Status:
mandatory.

OBJECT:

snmpInTotalSetVars { snmp 14 }

Syntax:
Counter

Definition:

The total number of MIB objects which have been altered successfully by the SNMP protocol entity as the result of receiving valid SNMP Set-Request PDUs.

Access:
read-only.

Status:
mandatory.

OBJECT:

snmpInGetRequests { snmp 15 }

Syntax:
Counter

Definition:

The total number of SNMP Get-Request PDUs which have been accepted and processed by the SNMP protocol entity.

Access:
read-only.

Status:
mandatory.

OBJECT:

snmpInGetNexts { snmp 16 }

Syntax:
Counter

Definition:

The total number of SNMP Get-Next PDUs which have been accepted and processed by the SNMP protocol entity.

Access:

read-only.

Status:

mandatory.

OBJECT:

snmpInSetRequests { snmp 17 }

Syntax:

Counter

Definition:

The total number of SNMP Set-Request PDUs which have been accepted and processed by the SNMP protocol entity.

Access:

read-only.

Status:

mandatory.

OBJECT:

snmpInGetResponses { snmp 18 }

Syntax:

Counter

Definition:

The total number of SNMP Get-Response PDUs which have been accepted and processed by the SNMP protocol entity.

Access:

read-only.

Status:

mandatory.

OBJECT:

snmpInTraps { snmp 19 }

Syntax:

Counter

Definition:

The total number of SNMP Trap PDUs which have been accepted and processed by the SNMP protocol entity.

Access:

read-only.

Status:

mandatory.

OBJECT:

snmpOutTooBigs { snmp 20 }

Syntax:

Counter

Definition:

The total number valid SNMP PDUs which were generated by the SNMP protocol entity and for which the value of the "ErrorStatus" component is "tooBig."

Access:

read-only.

Status:

mandatory.

OBJECT:

snmpOutNoSuchNames { snmp 21 }

Syntax:

Counter

Definition:

The total number valid SNMP PDUs which were generated by the SNMP protocol entity and for which the value of the "ErrorStatus" component is "noSuchName."

Access:
 read-only.

Status:
 mandatory.

OBJECT:

 snmpOutBadValues { snmp 22 }

Syntax:
 Counter

Definition:
 The total number valid SNMP PDUs which were generated by
 the SNMP protocol entity and for which the value of the
 "ErrorStatus" component is "badValue."

Access:
 read-only.

Status:
 mandatory.

OBJECT:

 snmpOutReadOnlys { snmp 23 }

Syntax:
 Counter

Definition:
 The total number valid SNMP PDUs which were generated by
 the SNMP protocol entity and for which the value of the
 "ErrorStatus" component is "readOnly."

Access:
 read-only.

Status:
 mandatory.

OBJECT:

 snmpOutGenErrs { snmp 24 }

Syntax:
Counter

Definition:
The total number valid SNMP PDUs which were generated by the SNMP protocol entity and for which the value of the "ErrorStatus" component is "genErr."

Access:
read-only.

Status:
mandatory.

OBJECT:

snmpOutGetRequests { snmp 25 }

Syntax:
Counter

Definition:
The total number of SNMP Get-Request PDUs which have been generated by the SNMP protocol entity.

Access:
read-only.

Status:
mandatory.

OBJECT:

snmpOutGetNexts { snmp 26 }

Syntax:
Counter

Definition:
The total number of SNMP Get-Next PDUs which have been generated by the SNMP protocol entity.

Access:
read-only.

Status:
mandatory.

OBJECT:

snmpOutSetRequests { snmp 27 }

Syntax:
Counter

Definition:
The total number of SNMP Set-Request PDUs which have been generated by the SNMP protocol entity.

Access:
read-only.

Status:
mandatory.

OBJECT:

snmpOutGetResponses { snmp 28 }

Syntax:
Counter

Definition:
The total number of SNMP Get-Response PDUs which have been generated by the SNMP protocol entity.

Access:
read-only.

Status:
mandatory.

OBJECT:

snmpOutTraps { snmp 29 }

Syntax:
Counter

Definition:

The total number of SNMP Trap PDUs which have been generated by the SNMP protocol entity.

Access:

read-only.

Status:

mandatory.

OBJECT:

snmpEnableAuthTraps { snmp 30 }

Syntax:

```
INTEGER {
    enabled(1),
    disabled(2)
}
```

Definition:

Indicates whether the SNMP agent process is configured to generate authentication-failure traps.

Access:

read-write.

Status:

mandatory.

6. Definitions

RFC1158-MIB

DEFINITIONS ::= BEGIN

IMPORTS

mgmt, OBJECT-TYPE, NetworkAddress, IPAddress,
Counter, Gauge, TimeTicks
FROM RFC1155-SMI;

mib-2 OBJECT IDENTIFIER ::= { mgmt 1 } -- MIB-II
-- (same prefix as MIB-I)

system OBJECT IDENTIFIER ::= { mib-2 1 }
interfaces OBJECT IDENTIFIER ::= { mib-2 2 }
at OBJECT IDENTIFIER ::= { mib-2 3 }

```
ip          OBJECT IDENTIFIER ::= { mib-2 4 }
icmp       OBJECT IDENTIFIER ::= { mib-2 5 }
tcp        OBJECT IDENTIFIER ::= { mib-2 6 }
udp        OBJECT IDENTIFIER ::= { mib-2 7 }
egp        OBJECT IDENTIFIER ::= { mib-2 8 }
-- cmot    OBJECT IDENTIFIER ::= { mib-2 9 }
transmission OBJECT IDENTIFIER ::= { mib-2 10 }
snmp       OBJECT IDENTIFIER ::= { mib-2 11 }

-- object types

-- the System group

sysDescr OBJECT-TYPE
    SYNTAX  DisplayString (SIZE (0..255))
    ACCESS  read-only
    STATUS  mandatory
    ::= { system 1 }

sysObjectID OBJECT-TYPE
    SYNTAX  OBJECT IDENTIFIER
    ACCESS  read-only
    STATUS  mandatory
    ::= { system 2 }

sysUpTime OBJECT-TYPE
    SYNTAX  TimeTicks
    ACCESS  read-only
    STATUS  mandatory
    ::= { system 3 }

sysContact OBJECT-TYPE
    SYNTAX  DisplayString (SIZE (0..255))
    ACCESS  read-write
    STATUS  mandatory
    ::= { system 4 }

sysName OBJECT-TYPE
    SYNTAX  DisplayString (SIZE (0..255))
    ACCESS  read-write
    STATUS  mandatory
    ::= { system 5 }

sysLocation OBJECT-TYPE
    SYNTAX  DisplayString (SIZE (0..255))
    ACCESS  read-only
    STATUS  mandatory
```



```
 ::= { system 6 }

sysServices OBJECT-TYPE
    SYNTAX  INTEGER (0..127)
    ACCESS  read-only
    STATUS  mandatory
    ::= { system 7 }

-- the Interfaces group

ifNumber OBJECT-TYPE
    SYNTAX  INTEGER
    ACCESS  read-only
    STATUS  mandatory
    ::= { interfaces 1 }

-- the Interfaces table

ifTable OBJECT-TYPE
    SYNTAX  SEQUENCE OF IfEntry
    ACCESS  read-only
    STATUS  mandatory
    ::= { interfaces 2 }

ifEntry OBJECT-TYPE
    SYNTAX  IfEntry
    ACCESS  read-only
    STATUS  mandatory
    ::= { ifTable 1 }

IfEntry ::= SEQUENCE {
    ifIndex
        INTEGER,
    ifDescr
        DisplayString,
    ifType
        INTEGER,
    ifMtu
        INTEGER,
    ifSpeed
        Gauge,
    ifPhysAddress
        OCTET STRING,
    ifAdminStatus
        INTEGER,
    ifOperStatus
        INTEGER,
```

```

    ifLastChange
        TimeTicks,
    ifInOctets
        Counter,
    ifInUcastPkts
        Counter,
    ifInNUcastPkts
        Counter,
    ifInDiscards
        Counter,
    ifInErrors
        Counter,
    ifInUnknownProtos
        Counter,
    ifOutOctets
        Counter,
    ifOutUcastPkts
        Counter,
    ifOutNUcastPkts
        Counter,
    ifOutDiscards
        Counter,
    ifOutErrors
        Counter,
    ifOutQLen
        Gauge,
    ifSpecific
        OBJECT IDENTIFIER
}

ifIndex OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS mandatory
    ::= { ifEntry 1 }

ifDescr OBJECT-TYPE
    SYNTAX DisplayString (SIZE (0..255))
    ACCESS read-only
    STATUS mandatory
    ::= { ifEntry 2 }

ifType OBJECT-TYPE
    SYNTAX INTEGER {
        other(1),                -- none of the
                                -- following
        regular1822(2),
        hdh1822(3),
    }

```

```

        ddn-x25(4),
        rfc877-x25(5),
        ethernet-csmacd(6),
        iso88023-csmacd(7),
        iso88024-tokenBus(8),
        iso88025-tokenRing(9),
        iso88026-man(10),
        starLan(11),
        proteon-10Mbit(12),
        proteon-80Mbit(13),
        hyperchannel(14),
        fddi(15),
        lapb(16),
        sdlc(17),
        t1-carrier(18),
        cept(19),          -- european
                          --equivalent of T-1
        basicISDN(20),
        primaryISDN(21),
                          -- proprietary
                          -- serial
        propPointToPointSerial(22),
        terminalServer-asyncPort(23),
        softwareLoopback(24),
        eon(25),          -- CLNP over IP
        ethernet-3Mbit(26),
        nsip(27),        -- XNS over IP
        slip(28)         -- generic SLIP
    }
    ACCESS read-only
    STATUS mandatory
    ::= { ifEntry 3 }

ifMtu OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS mandatory
    ::= { ifEntry 4 }

ifSpeed OBJECT-TYPE
    SYNTAX Gauge
    ACCESS read-only
    STATUS mandatory
    ::= { ifEntry 5 }

ifPhysAddress OBJECT-TYPE
    SYNTAX OCTET STRING
    ACCESS read-only

```

```
STATUS mandatory
 ::= { ifEntry 6 }

ifAdminStatus OBJECT-TYPE
SYNTAX INTEGER {
    up(1),          -- ready to pass packets
    down(2),
    testing(3)     -- in some test mode
}
ACCESS read-write
STATUS mandatory
 ::= { ifEntry 7 }

ifOperStatus OBJECT-TYPE
SYNTAX INTEGER {
    up(1),          -- ready to pass packets
    down(2),
    testing(3)     -- in some test mode
}
ACCESS read-only
STATUS mandatory
 ::= { ifEntry 8 }

ifLastChange OBJECT-TYPE
SYNTAX TimeTicks
ACCESS read-only
STATUS mandatory
 ::= { ifEntry 9 }

ifInOctets OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
 ::= { ifEntry 10 }

ifInUcastPkts OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
 ::= { ifEntry 11 }

ifInNUcastPkts OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
 ::= { ifEntry 12 }

ifInDiscards OBJECT-TYPE
```

```
        SYNTAX Counter
        ACCESS read-only
        STATUS mandatory
        ::= { ifEntry 13 }

ifInErrors OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { ifEntry 14 }

ifInUnknownProtos OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { ifEntry 15 }

ifOutOctets OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { ifEntry 16 }

ifOutUcastPkts OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { ifEntry 17 }

ifOutNUcastPkts OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { ifEntry 18 }

ifOutDiscards OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { ifEntry 19 }

ifOutErrors OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { ifEntry 20 }

ifOutQLen OBJECT-TYPE
```

```
        SYNTAX Gauge
        ACCESS read-only
        STATUS mandatory
        ::= { ifEntry 21 }

ifSpecific OBJECT-TYPE
    SYNTAX OBJECT IDENTIFIER
    ACCESS read-only
    STATUS mandatory
    ::= { ifEntry 22 }

nullSpecific OBJECT IDENTIFIER ::= { 0 0 }

-- the Address Translation group (deprecated)

atTable OBJECT-TYPE
    SYNTAX SEQUENCE OF AtEntry
    ACCESS read-write
    STATUS deprecated
    ::= { at 1 }

atEntry OBJECT-TYPE
    SYNTAX AtEntry
    ACCESS read-write
    STATUS deprecated
    ::= { atTable 1 }

AtEntry ::= SEQUENCE {
    atIfIndex
        INTEGER,
    atPhysAddress
        OCTET STRING,
    atNetAddress
        NetworkAddress
}

atIfIndex OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-write
    STATUS deprecated
    ::= { atEntry 1 }

atPhysAddress OBJECT-TYPE
    SYNTAX OCTET STRING
    ACCESS read-write
    STATUS deprecated
    ::= { atEntry 2 }
```

```
atNetAddress OBJECT-TYPE
    SYNTAX  NetworkAddress
    ACCESS  read-write
    STATUS  deprecated
    ::= { atEntry 3 }

-- the IP group

ipForwarding OBJECT-TYPE
    SYNTAX  INTEGER {
                gateway(1), -- entity forwards
                               -- datagrams
                host(2)     -- entity does NOT
                               -- forward datagrams
            }
    ACCESS  read-write
    STATUS  mandatory
    ::= { ip 1 }

ipDefaultTTL OBJECT-TYPE
    SYNTAX  INTEGER
    ACCESS  read-write
    STATUS  mandatory
    ::= { ip 2 }

ipInReceives OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    ::= { ip 3 }

ipInHdrErrors OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    ::= { ip 4 }

ipInAddrErrors OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    ::= { ip 5 }

ipForwDatagrams OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
```

```
 ::= { ip 6 }

ipInUnknownProtos OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { ip 7 }

ipInDiscards OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { ip 8 }

ipInDelivers OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { ip 9 }

ipOutRequests OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { ip 10 }

ipOutDiscards OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { ip 11 }

ipOutNoRoutes OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { ip 12 }

ipReasmTimeout OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS mandatory
    ::= { ip 13 }

ipReasmReqds OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
```



```
 ::= { ip 14 }

ipReasmOKs OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { ip 15 }

ipReasmFails OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { ip 16 }

ipFragOKs OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { ip 17 }

ipFragFails OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { ip 18 }

ipFragCreates OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { ip 19 }

-- the IP Interface table

ipAddrTable OBJECT-TYPE
    SYNTAX SEQUENCE OF IpAddrEntry
    ACCESS read-only
    STATUS mandatory
    ::= { ip 20 }

ipAddrEntry OBJECT-TYPE
    SYNTAX IpAddrEntry
    ACCESS read-only
    STATUS mandatory
    ::= { ipAddrTable 1 }

IpAddrEntry ::= SEQUENCE {
    ipAdEntAddr
```

```
        IpAddress,
        ipAdEntIfIndex
            INTEGER,
        ipAdEntNetMask
            IpAddress,
        ipAdEntBcastAddr
            INTEGER,
        ipAdEntReasmMaxSize
            INTEGER (0..65535)
    }

    ipAdEntAddr OBJECT-TYPE
        SYNTAX  IpAddress
        ACCESS  read-only
        STATUS  mandatory
        ::= { ipAddrEntry 1 }

    ipAdEntIfIndex OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory
        ::= { ipAddrEntry 2 }

    ipAdEntNetMask OBJECT-TYPE
        SYNTAX  IpAddress
        ACCESS  read-only
        STATUS  mandatory
        ::= { ipAddrEntry 3 }

    ipAdEntBcastAddr OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory
        ::= { ipAddrEntry 4 }

    ipAdEntReasmMaxSiz OBJECT-TYPE
        SYNTAX  INTEGER (0..65535)
        ACCESS  read-only
        STATUS  mandatory
        ::= { ipAddrEntry 5 }

    -- the IP Routing table

    ipRoutingTable OBJECT-TYPE
        SYNTAX  SEQUENCE OF IpRouteEntry
        ACCESS  read-write
        STATUS  mandatory
        ::= { ip 21 }
```

```
ipRouteEntry OBJECT-TYPE
    SYNTAX IpRouteEntry
    ACCESS read-write
    STATUS mandatory
    ::= { ipRoutingTable 1 }
```

```
IpRouteEntry ::= SEQUENCE {
    ipRouteDest
        IpAddress,
    ipRouteIfIndex
        INTEGER,
    ipRouteMetric1
        INTEGER,
    ipRouteMetric2
        INTEGER,
    ipRouteMetric3
        INTEGER,
    ipRouteMetric4
        INTEGER,
    ipRouteNextHop
        IpAddress,
    ipRouteType
        INTEGER,
    ipRouteProto
        INTEGER,
    ipRouteAge
        INTEGER,
    ipRouteMask
        IpAddress
}
```

```
ipRouteDest OBJECT-TYPE
    SYNTAX IpAddress
    ACCESS read-write
    STATUS mandatory
    ::= { ipRouteEntry 1 }
```

```
ipRouteIfIndex OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-write
    STATUS mandatory
    ::= { ipRouteEntry 2 }
```

```
ipRouteMetric1 OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-write
    STATUS mandatory
    ::= { ipRouteEntry 3 }
```

```
ipRouteMetric2 OBJECT-TYPE
    SYNTAX  INTEGER
    ACCESS  read-write
    STATUS  mandatory
    ::= { ipRouteEntry 4 }

ipRouteMetric3 OBJECT-TYPE
    SYNTAX  INTEGER
    ACCESS  read-write
    STATUS  mandatory
    ::= { ipRouteEntry 5 }

ipRouteMetric4 OBJECT-TYPE
    SYNTAX  INTEGER
    ACCESS  read-write
    STATUS  mandatory
    ::= { ipRouteEntry 6 }

ipRouteNextHop OBJECT-TYPE
    SYNTAX  IpAddress
    ACCESS  read-write
    STATUS  mandatory
    ::= { ipRouteEntry 7 }

ipRouteType OBJECT-TYPE
    SYNTAX  INTEGER {
        other(1),      -- none of the following
                       --
        invalid(2),   -- an invalidated route
                       --
        direct(3),    -- route to directly
                       -- connected
                       -- (sub-)network
                       --
        remote(4)     -- route to a non-local
                       -- host/network/
                       -- sub-network
    }
    ACCESS  read-write
    STATUS  mandatory
    ::= { ipRouteEntry 8 }

ipRouteProto OBJECT-TYPE
    SYNTAX  INTEGER {
        other(1),      -- none of the following
                       --
                       -- non-protocol
                       -- information
    }
```

```

                                -- e.g., manually
                                -- configured entries
                                local(2),
                                -- set via a network
                                -- management protocol
                                netmgmt(3),
                                -- obtained via ICMP,
                                -- e.g., Redirect
                                icmp(4),
                                -- the following are
                                -- gateway routing
                                -- protocols
                                egp(5),
                                ggp(6),
                                hello(7),
                                rip(8),
                                is-is(9),
                                es-is(10),
                                ciscoIgrp(11),
                                bbnSpfIgp(12),
                                ospf(13)
                                bgp(14)
                                }
                                ACCESS read-only
                                STATUS mandatory
                                ::= { ipRouteEntry 9 }

ipRouteAge OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-write
    STATUS mandatory
    ::= { ipRouteEntry 10 }

ipRouteMask OBJECT-TYPE
    SYNTAX IpAddress
    ACCESS read-write
    STATUS mandatory
    ::= { ipRouteEntry 11 }

-- the IP Address Translation tables

ipNetToMediaTable OBJECT-TYPE
    SYNTAX SEQUENCE OF IpNetToMediaEntry
    ACCESS read-write
    STATUS mandatory
    ::= { ip 22 }

ipNetToMediaEntry OBJECT-TYPE

```

```

        SYNTAX IpNetToMediaEntry
        ACCESS read-write
        STATUS mandatory
        ::= { ipNetToMediaTable 1 }

IpNetToMediaEntry ::= SEQUENCE {
    ipNetToMediaIfIndex
        INTEGER,
    ipNetToMediaPhysAddress
        OCTET STRING,
    ipNetToMediaNetAddress
        IpAddress,
    ipNetToMediaType
        INTEGER
}

ipNetToMediaIfIndex OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-write
    STATUS mandatory
    ::= { ipNetToMediaEntry 1 }

ipNetToMediaPhysAddress OBJECT-TYPE
    SYNTAX OCTET STRING
    ACCESS read-write
    STATUS mandatory
    ::= { ipNetToMediaEntry 2 }

ipNetToMediaNetAddress OBJECT-TYPE
    SYNTAX IpAddress
    ACCESS read-write
    STATUS mandatory
    ::= { ipNetToMediaEntry 3 }

ipNetToMediaType OBJECT-TYPE
    SYNTAX INTEGER {
        other(1), -- none of the following
        invalid(2), -- an invalidated mapping
        dynamic(3), -- connected (sub-)network
        static(4)
    }
    ACCESS read-write
    STATUS mandatory
    ::= { ipNetToMediaEntry 4 }

```

```
-- the ICMP group

icmpInMsgs OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { icmp 1 }

icmpInErrors OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { icmp 2 }

icmpInDestUnreachs OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { icmp 3 }

icmpInTimeExcds OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { icmp 4 }

icmpInParmProbs OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { icmp 5 }

icmpInSrcQuenchs OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { icmp 6 }

icmpInRedirects OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { icmp 7 }

icmpInEchos OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
```

```
 ::= { icmp 8 }

icmpInEchoReps OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { icmp 9 }

icmpInTimestamps OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { icmp 10 }

icmpInTimestampReps OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { icmp 11 }

icmpInAddrMasks OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { icmp 12 }

icmpInAddrMaskReps OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { icmp 13 }

icmpOutMsgs OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { icmp 14 }

icmpOutErrors OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { icmp 15 }

icmpOutDestUnreachs OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
```



```
 ::= { icmp 16 }

icmpOutTimeExcds OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { icmp 17 }

icmpOutParmProbs OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { icmp 18 }

icmpOutSrcQuenchs OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { icmp 19 }

icmpOutRedirects OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { icmp 20 }

icmpOutEchos OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { icmp 21 }

icmpOutEchoReps OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { icmp 22 }

icmpOutTimestamps OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { icmp 23 }

icmpOutTimestampReps OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
```

```
 ::= { icmp 24 }

icmpOutAddrMasks OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { icmp 25 }

icmpOutAddrMaskReps OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { icmp 26 }

-- the TCP group

tcpRtoAlgorithm OBJECT-TYPE
    SYNTAX INTEGER {
        other(1), -- none of the following
        constant(2), -- a constant rto
        rsre(3), -- MIL-STD-1778,
        -- Appendix B
        vanj(4) -- Van Jacobson's
        -- algorithm
    }
    ACCESS read-only
    STATUS mandatory
    ::= { tcp 1 }

tcpRtoMin OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS mandatory
    ::= { tcp 2 }

tcpRtoMax OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS mandatory
    ::= { tcp 3 }

tcpMaxConn OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS mandatory
    ::= { tcp 4 }
```

```
tcpActiveOpens OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { tcp 5 }

tcpPassiveOpens OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { tcp 6 }

tcpAttemptFails OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { tcp 7 }

tcpEstabResets OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { tcp 8 }

tcpCurrEstab OBJECT-TYPE
    SYNTAX Gauge
    ACCESS read-only
    STATUS mandatory
    ::= { tcp 9 }

tcpInSegs OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { tcp 10 }

tcpOutSegs OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { tcp 11 }

tcpRetransSegs OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { tcp 12 }
```

```
-- the TCP connections table

tcpConnTable OBJECT-TYPE
    SYNTAX SEQUENCE OF TcpConnEntry
    ACCESS read-only
    STATUS mandatory
    ::= { tcp 13 }

tcpConnEntry OBJECT-TYPE
    SYNTAX TcpConnEntry
    ACCESS read-only
    STATUS mandatory
    ::= { tcpConnTable 1 }

TcpConnEntry ::= SEQUENCE {
    tcpConnState
        INTEGER,
    tcpConnLocalAddress
        IpAddress,
    tcpConnLocalPort
        INTEGER (0..65535),
    tcpConnRemAddress
        IpAddress,
    tcpConnRemPort
        INTEGER (0..65535)
}

tcpConnState OBJECT-TYPE
    SYNTAX INTEGER {
        closed(1),
        listen(2),
        synSent(3),
        synReceived(4),
        established(5),
        finWait1(6),
        finWait2(7),
        closeWait(8),
        lastAck(9),
        closing(10),
        timeWait(11)
    }
    ACCESS read-only
    STATUS mandatory
    ::= { tcpConnEntry 1 }

tcpConnLocalAddress OBJECT-TYPE
    SYNTAX IpAddress
    ACCESS read-only
```

```
STATUS mandatory
 ::= { tcpConnEntry 2 }

tcpConnLocalPort OBJECT-TYPE
SYNTAX INTEGER (0..65535)
ACCESS read-only
STATUS mandatory
 ::= { tcpConnEntry 3 }

tcpConnRemAddress OBJECT-TYPE
SYNTAX IpAddress
ACCESS read-only
STATUS mandatory
 ::= { tcpConnEntry 4 }

tcpConnRemPort OBJECT-TYPE
SYNTAX INTEGER (0..65535)
ACCESS read-only
STATUS mandatory
 ::= { tcpConnEntry 5 }

-- additional TCP variables

tcpInErrs OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
 ::= { tcp 14 }

tcpOutRsts OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
 ::= { tcp 15 }

-- the UDP group

udpInDatagrams OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
 ::= { udp 1 }

udpNoPorts OBJECT-TYPE
SYNTAX Counter
ACCESS read-only
STATUS mandatory
```

```
 ::= { udp 2 }

udpInErrors OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { udp 3 }

udpOutDatagrams OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { udp 4 }

-- the UDP listener table

udpTable OBJECT-TYPE
    SYNTAX SEQUENCE OF UdpEntry
    ACCESS read-only
    STATUS mandatory
    ::= { udp 5 }

udpEntry OBJECT-TYPE
    SYNTAX UdpEntry
    ACCESS read-only
    STATUS mandatory
    ::= { udpTable 1 }

UdpEntry ::= SEQUENCE {
    udpLocalAddress
        IpAddress,
    udpLocalPort
        INTEGER (0..65535)
}

udpLocalAddress OBJECT-TYPE
    SYNTAX IpAddress
    ACCESS read-only
    STATUS mandatory
    ::= { udpEntry 1 }

udpLocalPort OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    ACCESS read-only
    STATUS mandatory
    ::= { udpEntry 2 }
```

```
-- the EGP group

egpInMsgs OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { egp 1 }

egpInErrors OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { egp 2 }

egpOutMsgs OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { egp 3 }

egpOutErrors OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { egp 4 }

-- the EGP Neighbor table

egpNeighTable OBJECT-TYPE
    SYNTAX SEQUENCE OF EgpNeighEntry
    ACCESS read-only
    STATUS mandatory
    ::= { egp 5 }

egpNeighEntry OBJECT-TYPE
    SYNTAX EgpNeighEntry
    ACCESS read-only
    STATUS mandatory
    ::= { egpNeighTable 1 }

EgpNeighEntry ::= SEQUENCE {
    egpNeighState
        INTEGER,
    egpNeighAddr
        IpAddress,
    egpNeighAs
        INTEGER,
    egpNeighInMsgs
```

```
        Counter,
    egpNeighInErrs
        Counter,
    egpNeighOutMsgs
        Counter,
    egpNeighOutErrs
        Counter,
    egpNeighInErrMsgs
        Counter,
    egpNeighOutErrMsgs
        Counter,
    egpNeighStateUps
        Counter,
    egpNeighStateDowns
        Counter,
    egpNeighIntervalHello
        INTEGER,
    egpNeighIntervalPoll
        INTEGER,
    egpNeighMode
        INTEGER,
    egpNeighEventTrigger
        INTEGER
}

egpNeighState OBJECT-TYPE
    SYNTAX  INTEGER {
                idle(1),
                acquisition(2),
                down(3),
                up(4),
                cease(5)
            }
    ACCESS  read-only
    STATUS  mandatory
    ::= { egpNeighEntry 1 }

egpNeighAddr OBJECT-TYPE
    SYNTAX  IpAddress
    ACCESS  read-only
    STATUS  mandatory
    ::= { egpNeighEntry 2 }

egpNeighAs OBJECT-TYPE
    SYNTAX  INTEGER
    ACCESS  read-only
    STATUS  mandatory
    ::= { egpNeighEntry 3 }
```



```
egpNeighInMsgs OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { egpNeighEntry 4 }

egpNeighInErrs OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { egpNeighEntry 5 }

egpNeighOutMsgs OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { egpNeighEntry 6 }

egpNeighOutErrs OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { egpNeighEntry 7 }

egpNeighInErrMsgs OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { egpNeighEntry 8 }

egpNeighOutErrMsgs OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { egpNeighEntry 9 }

egpNeighStateUps OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { egpNeighEntry 10 }

egpNeighStateDowns OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { egpNeighEntry 11 }
```

```
egpNeighIntervalHello OBJECT-TYPE
    SYNTAX  INTEGER
    ACCESS  read-only
    STATUS  mandatory
    ::= { egpNeighEntry 12 }

egpNeighIntervalPoll OBJECT-TYPE
    SYNTAX  INTEGER
    ACCESS  read-only
    STATUS  mandatory
    ::= { egpNeighEntry 13 }

egpNeighMode OBJECT-TYPE
    SYNTAX  INTEGER {
        active(1),
        passive(2)
    }
    ACCESS  read-only
    STATUS  mandatory
    ::= { egpNeighEntry 14 }

egpNeighEventTrigger OBJECT-TYPE
    SYNTAX  INTEGER {
        start(1),
        stop(2)
    }
    ACCESS  read-write
    STATUS  mandatory
    ::= { egpNeighEntry 15 }

-- additional EGP variables

egpAs OBJECT-TYPE
    SYNTAX  INTEGER
    ACCESS  read-only
    STATUS  mandatory
    ::= { egp 6 }

-- the Transmission group (empty at present)

-- the SNMP group

snmpInPkts OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    ::= { snmp 1 }
```

```
snmpOutPkts OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { snmp 2 }

snmpInBadVersions OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { snmp 3 }

snmpInBadCommunityNames OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { snmp 4 }

snmpInBadCommunityUses OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { snmp 5 }

snmpInASNParseErrs OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { snmp 6 }

snmpInBadTypes OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { snmp 7 }

snmpInTooBigs OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { snmp 8 }

snmpInNoSuchNames OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { snmp 9 }
```

```
snmpInBadValues OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { snmp 10 }

snmpInReadOnlys OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { snmp 11 }

snmpInGenErrs OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { snmp 12 }

snmpInTotalReqVars OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { snmp 13 }

snmpInTotalSetVars OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { snmp 14 }

snmpInGetRequests OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { snmp 15 }

snmpInGetNexts OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { snmp 16 }

snmpInSetRequests OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { snmp 17 }
```

```
snmpInGetResponses OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { snmp 18 }

snmpInTraps OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { snmp 19 }

snmpOutTooBigs OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { snmp 20 }

snmpOutNoSuchNames OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { snmp 21 }

snmpOutBadValues OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { snmp 22 }

snmpOutReadOnlys OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { snmp 23 }

snmpOutGenErrs OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { snmp 24 }

snmpOutGetRequests OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { snmp 25 }
```

```

snmpOutGetNexts OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { snmp 26 }

snmpOutSetRequests OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { snmp 27 }

snmpOutGetResponses OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { snmp 28 }

snmpOutTraps OBJECT-TYPE
    SYNTAX Counter
    ACCESS read-only
    STATUS mandatory
    ::= { snmp 29 }

snmpEnableAuthTraps OBJECT-TYPE
    SYNTAX INTEGER {
        enabled(1),
        disabled(2)
    }
    ACCESS read-write
    STATUS mandatory
    ::= { snmp 30 }

```

END

7. Identification of OBJECT instances for use with the SNMP

The names for all object types in the MIB are defined explicitly either in the Internet-standard MIB or in other documents which conform to the naming conventions of the SMI. The SMI requires that conformant management protocols define mechanisms for identifying individual instances of those object types for a particular network element.

Each instance of any object type defined in the MIB is identified in SNMP operations by a unique name called its "variable name." In general, the name of an SNMP variable is an OBJECT IDENTIFIER of the form x.y, where x is the name of a non-aggregate object type defined

in the MIB and *y* is an OBJECT IDENTIFIER fragment that, in a way specific to the named object type, identifies the desired instance.

This naming strategy admits the fullest exploitation of the semantics of the powerful SNMP get-next operator, because it assigns names for related variables so as to be contiguous in the lexicographical ordering of all variable names known in the MIB.

The type-specific naming of object instances is defined below for a number of classes of object types. Instances of an object type to which none of the following naming conventions are applicable are named by OBJECT IDENTIFIERS of the form *x.0*, where *x* is the name of said object type in the MIB definition.

For example, suppose one wanted to identify an instance of the variable `sysDescr`. The object class for `sysDescr` is:

```
iso org dod internet mgmt mib system sysDescr
 1  3  6      1      2    1    1      1
```

Hence, the object type, *x*, would be 1.3.6.1.2.1.1.1 to which is appended an instance sub-identifier of 0. That is, 1.3.6.1.2.1.1.1.0 identifies the one and only instance of `sysDescr`.

7.1. ifTable Object Type Names

The name of a subnetwork interface, *s*, is the OBJECT IDENTIFIER value of the form *i*, where *i* has the value of that instance of the `ifIndex` object type associated with *s*. For each object type, *t*, for which the defined name, *n*, has a prefix of `ifEntry`, an instance, *i*, of *t* is named by an OBJECT IDENTIFIER of the form *n.s*, where *s* is the name of the subnetwork interface about which *i* represents information.

For example, suppose one wanted to identify the instance of the variable `ifType` associated with interface 2. Accordingly, `ifType.2` would identify the desired instance.

7.2. atTable Object Type Names

The name of an address translation entry, *x*, is an OBJECT IDENTIFIER of the form *s.1.a.b.c.d*, such that *s* is the value of that instance of the `atIfIndex` object type associated with *x*, the subidentifier "1" signifies the translation of an IP protocol address, and *a.b.c.d* is the IP address value (in the familiar "dot" notation) of that instance of the `atNetAddress` object type associated with *x*.

For each object type, *t*, for which the defined name, *n*, has a prefix of `atEntry`, an instance, *i*, of *t* is named by an OBJECT IDENTIFIER of

the form *n.y*, where *y* is the name of the address translation entry about which *i* represents information.

For example, suppose one wanted to find the physical address of an entry in the address translation table (ARP cache) associated with an IP address of 89.1.1.42 and interface 3. Accordingly, `atPhysAddress.3.1.89.1.1.42` would identify the desired instance.

7.3. ipAddrTable Object Type Names

The name of an IP-addressable network element, *x*, is the OBJECT IDENTIFIER of the form *a.b.c.d* such that *a.b.c.d* is the value (in the familiar "dot" notation) of that instance of the `ipAdEntAddr` object type associated with *x*.

For each object type, *t*, for which the defined name, *n*, has a prefix of `ipAddrEntry`, an instance, *i*, of *t* is named by an OBJECT IDENTIFIER of the form *n.y*, where *y* is the name of the IP-addressable network element about which *i* represents information.

For example, suppose one wanted to find the network mask of an entry in the IP interface table associated with an IP address of 89.1.1.42. Accordingly, `ipAdEntNetMask.89.1.1.42` would identify the desired instance.

At the option of the agent, multiple entries for the same IP address may be visible. To realize this, the agent, while required to return a single entry for an IP address, *x*, of the form *n.y*, may also return information about other entries for the same IP address using the form *n.y.z*, where *z* is a implementation-dependent small, non-negative integer. It is strongly recommended that the value of *z* correspond to the value of `ipAddrIfIndex` for that entry.

7.4. ipRoutingTable Object Type Names

The name of an IP route, *x*, is the OBJECT IDENTIFIER of the form *a.b.c.d* such that *a.b.c.d* is the value (in the familiar "dot" notation) of that instance of the `ipRouteDest` object type associated with *x*.

For each object type, *t*, for which the defined name, *n*, has a prefix of `ipRoutingEntry`, an instance, *i*, of *t* is named by an OBJECT IDENTIFIER of the form *n.y*, where *y* is the name of the IP route about which *i* represents information.

For example, suppose one wanted to find the next hop of an entry in the IP routing table associated with the destination of 89.1.1.42. Accordingly, `ipRouteNextHop.89.1.1.42` would identify the desired

instance.

At the option of the agent, multiple routes to the same destination may be visible. To realize this, the agent, while required to return a single entry for an IP route, *x*, of the form *n.y*, may also return information about other routes to the same destination using the form *n.y.z*, where *z* is a implementation-dependent small, non-negative integer.

7.5. ipNetToMediaTable Object Type Names

The name of a cached IP address, *x*, is an OBJECT IDENTIFIER of the form *s.a.b.c.d*, such that *s* is the value of that instance of the `ipNetToMediaIfIndex` object type associated with the entry and *a.b.c.d* is the value (in the familiar "dot" notation) of the `ipNetToMediaNetAddress` object type associated with *x*.

For each object type, *t*, for which the defined name, *n*, has a prefix of `ipNetToMediaEntry`, an instance, *i*, of *t* is named by an OBJECT IDENTIFIER of the form *n.y*, where *y* is the name of the cached IP address about which *i* represents information.

For example, suppose one wanted to find the media address of an entry in the address translation table associated with a IP address of 192.52.180.1 and interface 3. Accordingly, `ipNetToMediaPhysAddress.3.192.52.180.1` would identify the desired instance.

7.6. tcpConnTable Object Type Names

The name of a TCP connection, *x*, is the OBJECT IDENTIFIER of the form *a.b.c.d.e.f.g.h.i.j* such that *a.b.c.d* is the value (in the familiar "dot" notation) of that instance of the `tcpConnLocalAddress` object type associated with *x* and such that *f.g.h.i* is the value (in the familiar "dot" notation) of that instance of the `tcpConnRemoteAddress` object type associated with *x* and such that *e* is the value of that instance of the `tcpConnLocalPort` object type associated with *x* and such that *j* is the value of that instance of the `tcpConnRemotePort` object type associated with *x*.

For each object type, *t*, for which the defined name, *n*, has a prefix of `tcpConnEntry`, an instance, *i*, of *t* is named by an OBJECT IDENTIFIER of the form *n.y*, where *y* is the name of the TCP connection about which *i* represents information.

For example, suppose one wanted to find the state of a TCP connection between the local address of 89.1.1.42 on TCP port 21 and the remote address of 10.0.0.51 on TCP port 2059. Accordingly,

tcpConnState.89.1.1.42.21.10.0.0.51.2059 would identify the desired instance.

7.7. udpTable Object Type Names

The name of a UDP listener, *x*, is the OBJECT IDENTIFIER of the form a.b.c.d.e. such that a.b.c.d is the value (in the familiar "dot" notation) of that instance of the udpLocalAddress object type associated with *x* and such that e is the value of that instance of the udpLocalPort object type associated with *x*.

For each object type, *t*, for which the defined name, *n*, has a prefix of udpEntry, an instance, *i*, of *t* is named by an OBJECT IDENTIFIER of the form n.y, where *y* is the name of the UDP listener about which *i* represents information.

For example, suppose one wanted to determine if a UDP listener was present at the local address of 89.1.1.42 on UDP port 21. Accordingly, a successful retrieval of either udpLocalAddress.89.1.1.42.21 or udpLocalPort.89.1.1.42.21 would indicate this.

7.8. egpNeighTable Object Type Names

The name of an EGP neighbor, *x*, is the OBJECT IDENTIFIER of the form a.b.c.d such that a.b.c.d is the value (in the familiar "dot" notation) of that instance of the egpNeighAddr object type associated with *x*.

For each object type, *t*, for which the defined name, *n*, has a prefix of egpNeighEntry, an instance, *i*, of *t* is named by an OBJECT IDENTIFIER of the form n.y, where *y* is the name of the EGP neighbor about which *i* represents information.

For example, suppose one wanted to find the neighbor state for the IP address of 89.1.1.42. Accordingly, egpNeighState.89.1.1.42 would identify the desired instance.

8. Acknowledgements

This document was produced by the SNMP Working Group:

Karl Auerbach, Epilogue Technology
David Bridgham, Epilogue Technology
Brian Brown, Synoptics
John Burrell, Wellfleet
Jeffrey D. Case, University of Tennessee at Knoxville
James R. Davin, MIT-LCS

Mark S. Fedor, PSI, Inc.
Stan Froyd, ACC
Satish Joshi, Synoptics
Ken Key, University of Tennessee at Knoxville
Gary Malkin, Proteon
Randy Mayhew, University of Tennessee at Knoxville
Keith McCloghrie, Hughes LAN Systems
Marshall T. Rose, PSI, Inc. (chair)
Greg Satz, cisco
Martin Lee Schoffstall, PSI, Inc.
Bob Stewart, Xyplex
Geoff Thompson, Synoptics
Bill Versteeg, Network Research Corporation
Wengyik Yeong, PSI, Inc.

In addition, the comments of the following individuals are also
acknowledged:

Craig A. Finseth, Minnesota Supercomputer Center, Inc.
Jeffrey C. Honig, Cornell University Theory Center
Philip R. Karn, Bellcore
David Waitzman, BBN

9. References

- [1] Cerf, V., "IAB Recommendations for the Development of Internet Network Management Standards", RFC 1052, IAB, April 1988.
- [2] Rose, M., and K. McCloghrie, "Structure and Identification of Management Information for TCP/IP-based internets", RFC 1065, TWG, August 1988.
- [3] McCloghrie K., and M. Rose, "Management Information Base for Network Management of TCP/IP-based internets", RFC 1066, TWG, August 1988.
- [4] Cerf, V., "Report of the Second Ad Hoc Network Management Review Group", RFC 1109, IAB, August 1989.
- [5] Case, J., Fedor, M., Schoffstall, M., and J. Davin, "A Simple Network Management Protocol (SNMP)", RFC 1098, University of Tennessee at Knoxville, NYSERNet, Inc., Rensselaer Polytechnic Institute, MIT Laboratory for Computer Science, April 1989.
- [6] Warrier, U., and L. Besaw, "Common Management Information Services and Protocol over TCP/IP (CMOT)", RFC 1095, Unisys Corporation, Hewlett-Packard, April 1989.

- [7] Postel, J., "Telnet Protocol Specification", RFC 854, USC/Information Sciences Institute, May 1983.
- [8] Satz, G., "Experimental MIB Objects for the CLNP", Internet Working Group Request for Comments draft. Network Information Center, SRI International, Menlo Park, California, (in preparation).
- [9] Information processing systems - Open Systems Interconnection, "Specification of Abstract Syntax Notation One (ASN.1)", International Organization for Standardization, International Standard 8824, December 1987.
- [10] Information processing systems - Open Systems Interconnection, "Specification of Basic Encoding Rules for Abstract Notation One (ASN.1)", International Organization for Standardization. International Standard 8825, December 1987.
- [11] Jacobson, V., "Congestion Avoidance and Control", SIGCOMM 1988, Stanford, California.
- [12] Hagens, R., Hall, N., and M. Rose, "Use of the Internet as a subnetwork for experimentation with the OSI network layer", February, 1989.
- [13] Rose, M., and K. McCloghrie, "Structure and Identification of Management Information for TCP/IP-based Internets", RFC 1155, Performance Systems International and Hughes LAN Systems, May 1990.
- [14] Case, J., Fedor, M., Schoffstall, M., and J. Davin, "The Simple Network Management Protocol", RFC 1157, University of Tennessee at Knoxville, Performance Systems International, Performance Systems International, and the MIT Laboratory for Computer Science, May 1990.

10. Security Considerations

Security issues are not discussed in this memo.

11. Author's Address:

Marshall T. Rose
PSI, Inc.
PSI California Office
P.O. Box 391776
Mountain View, CA 94039

Phone: (415) 961-3380

Email: mrose@PSI.COM