Network Working Group
Request For Comment: 1024

C.Partridge
BBN/NNSC
G. Trewitt
Stanford
October 1987

HEMS VARIABLE DEFINITIONS

STATUS OF THIS MEMO

This memo assigns instruction codes, defines object formats and object semantics for use with the High-Level Monitoring and Control Language, defined in RFC-1023.

This memo is provisional and the definitions are subject to change. Readers should confirm that they have the most recent version of the memo.

The authors assume a working knowledge of the ISO data encoding standard, ASN.1, and a general understanding of the IP protocol suite.

Distribution of this memo is unlimited.

INTRODUCTION

In other memos [RFC-1021, RFC-1022] the authors have described a general system for monitoring and controlling network entities; this system is called the High-Level Entity Management System (HEMS). This system permits applications to read and write values in remote entities which support a simple query processor.

In this memo we standardize the language instruction codes, the objects which can be read or written, and the meanings of any constants stored in the objects. There are three parts to this standardization: (1) the assignment of an ASN.1 tag to each value, (2) the definition of the external representation of the value (e.g., INTEGER, OCTETSTRING, etc.), and (3) the definition of the meaning, or semantics of a value (e.g., what types of packets a particular packet counter actually tracks).

This definition is provisional, and the authors hope that it will be expanded and improved as the community becomes more experienced with HEMS. Readers with suggestions for additional object definitions, or improved definitions are encouraged to contact the authors.

MESSAGE FORMATS

All HEMS values are conveyed between applications and entities using the High-Level Entity Management Protocol (HEMP) specified in RFC-1022. All values specified in this memo are passed in the data sections of HEMP messages. For all message types, the data section is a SEQUENCE of objects. For requests, these objects are operations and their operands. Replies contain a sequence of objects retrieved by a request. Events contain an initial event object followed by an optional number of objects related to the event.

Messages conforming to this memo should set the link field in the HEMP CommonHeader to 1, to indicate version 1 of HEMS. The resourceId field should be set to NULL.

CONTROL LANGUAGE INSTRUCTIONS

The HEMS Monitoring and Control Language defines a suite of operations which the query processor must be able to perform. These operations and their operands are ASN.1 objects which are passed to the query processor over a network connection. The operations and operands are sent in postfix form (the operation follows the operands). Operands are pushed onto a stack and are processed when the operation is encountered.

To ensure that operations are easily recognized in the input stream, they are all encoded in a single application-specific type. This type is shown below.

```
Operation ::= [APPLICATION 1] IMPLICIT INTEGER {
        reserved(0), get(1) begin(2), end(3),
        get-match(4), get-attributes(5),
        get-attributes-match(6), get-range(7),
        set(8), set-match(9)
    }
```

When the query processor encounters an Operation object it consults the value to determine which operation is to be done (e.g., GET).

GENERAL COMMENTS ON OBJECTS STORED IN ENTITIES

The High-Level Monitoring and Control Language requires the object space to have a tree-shaped type space. Locating a particular object requires identifying that section of the tree in which the object resides. (A more detailed explanation of the scheme is given in RFC-1023).

This memo defines a universal type space. A subset of this type space is expected to be an appropriate type space for any entity (e.g., a gateway or a multi-user host). The type space is divided into required and optional portions. Implementors should implement the required portion of the type space plus that part of the optional type space which is appropriate for their particular entity.

One problem with defining a universal type space is that certain interesting objects are not universal, but are instead very machine specific (for example, status registers on specialized hardware). To allow implementors to retrieve such implementation-specific objects using the HEMS system, a special APPLICATION type is reserved for non-standard values.

Putting objects in ASN.1 form implies an ability to map to and from ASN.1 format. One of the design goals of this system has been to minimize the amount of ASN.1 compilation required by the query processor to reduce the expense of processing queries at entities. (This implies a certain willingness to force the applications querying entities to be more powerful). We expect that most of the complex mapping will be done when objects are read; most writable objects have a simple format (e.g., an INTEGER, or OCTETSTRING). As a result, we have made a heavy use of the ASN.1 SET type, which allows values to be presented in any order. Applications which require particular fields in an object may use the template structure to specify particular fields to be retrieved, but this still permits the query processor to return the fields in whatever order is convenient.

In addition to ease the problems of ASN.1 compilation, query processors are not required to reduce an INTEGER to the minimum number of octets as specified in ASN.1. Applications should be prepared to receive INTEGERs which have leading octets with all zeros or ones.

More generally, a design goal of HEMS was to try to limit the data processing done at the entity, and to place the burden of data reduction on the querying application. As a result, the objects presented here are typically counters, or values which the entity has to compute already. Object definitions which require the entity to do data reduction are not supported, although consideration might be given to making them optionally available.

Finally, HEMS is required to support access by multiple network management centers or applications. This constraint has some important consequences. First, the SET operation cannot be applied to any Counter, since changing the value of a Counter may impair data acquisition by other centers. More generally, there are questions

about competing or clashing SET requests from management centers. Currently HEMS does not provide any facilities for protecting against such requests. If such facilities become necessary, the authors envision the enhancement of the object definitions to incorporate the idea of "owned" objects.

READING THE OBJECT DEFINITIONS

Most of the rest of this memo is devoted to ennumerating the objects managed by the query processor. Many of these objects are dictionaries, objects which reference other objects. Defining dictionaries requires that we specify the class of objects they reference.

Most significant objects, such as packet counts, reside at the leaves of the object data tree. They need to be carefully defined to ensure that their meaning is consistent across all HEMS implementations. These values are defined using the following format:

OBJECT: This is the name of the object.

Type: This is the ASN.1 type of the object.

Definition: The meaning of the data the object contains. Implementations should ensure that their instance of the object fulfills this definition since an important feature of HEMS is that objects have consistent meaning across all machines. It is better not to implement an object than to abuse its definition.

Notes: An optional section of the definition which is used to discuss issues not covered in other sections of this specification.

Object Status: An optional section of the definition which is used to indicate whether the object is required of all HEMS implementations, encouraged of HEMS implementations or simply considered useful. Currently, there are four levels of status:

Required: The object is felt to provide critical information and must be included in a fully conforming HEMS implementation.

Required On Condition: The object is felt to provide critical information about an optional

feature of an IP entity (for example, support of the Transmission Control Protocol). The object is required if the feature is implemented in the entity.

Encouraged: The object is felt to provide very useful management information and implementors are encouraged to implement it.

Defined: The object may be useful and has been defined so that all implementations of the object are consistent.

If the object status is not specified, the object should be considered required. If the parent dictionary is optional, then the object should be considered required if the parent dictionary is supported.

Operations on Object: The definition of how each monitoring and control operation acts on the object. Many operations have the same effect on almost all values, so some default definitions are presented here. In the absence of an operation specification, implementors should use the default operations defined here.

BEGIN: The default is for BEGIN to be defined for dictionaries, and an error if performed on leaf objects in the tree.

CREATE: The default is that CREATE is undefined.

DELETE: The default is that DELETE is undefined.

END: END is a stack operation and is defined for all objects. Note that END may fail if there is no object on the stack.

GET-ATTRIBUTES: The default is that GET-ATTRIBUTES is defined on the contents of all dictionaries specified in this memo. The text description attributes are optional for values defined in this memo, but are required for implementation-specific objects. Any descriptions of object listed in this memo should cite this memo. GET-ATTRIBUTES must be supported on all entity-specific values. GET-ATTRIBUTES returns a Attributes object, which is defined in the well-known types section below.

GET-ATTRIBUTES-MATCH: The default is that GET-ATTRIBUTES-MATCH is optionally defined on any object which supports GET-MATCH, and is an error otherwise. The rules for attributes returned by GET-ATTRIBUTES-MATCH are the same as those for GET-ATTRIBUTES.

GET: The default definition of GET is to emit the operand specified is a leaf object, and if the operand is a dictionary, to recursively GET the entire dictionary and its subdictionaries.

GET-MATCH: Unless otherwise specified, GET-MATCH is not supported on an object.

GET-RANGE: Unless otherwise specified, GET-RANGE is not supported on an object.

SET: Unless otherwise specified, SET is not supported on an object.

SET-MATCH: Unless otherwise specified, SET-MATCH is not supported on an object.

ATTRIBUTES

HEMS requires that remote applications be able to discover the meaning of an object by querying the entity in which the object is stored. This is done through use of the GET-ATTRIBUTES operator, which causes an Attributes object to be returned to the application. The Attributes object is described below.

```
Attributes ::= [APPLICATION 2] IMPLICIT SEQUENCE {
    tagASN1       [0] IMPLICIT INTEGER,
    valueFormat   [1] IMPLICIT INTEGER,
    longDesc      [2] IMPLICIT IA5String OPTIONAL,
    shortDesc     [3] IMPLICIT IA5String OPTIONAL,
    unitsDesc     [4] IMPLICIT IA5String OPTIONAL,
    precision     [5] IMPLICIT INTEGER OPTIONAL,
    properties    [6] IMPLICIT BITSTRING OPTIONAL,
}
```

The meanings of the various attributes are given below.

tagASN1: The ASN.1 tag for this object. This attribute is required.

valueFormat: The underlying ASN.1 type of the object (e.g., SEQUENCE, or OCTETSTRING). This attribute is required.

longDesc: A potentially lengthy text description which fully defines the object. This attribute is optional for objects defined in this memo and required for entity-specific objects.

shortDesc: A short mnemonic string of less than 15 octets which is suitable for labelling the value on a display. This attribute is optional.

unitsDesc: A short string used for integer values to indicate the units in which the value is measured (e.g. "ms", "sec", "packets", etc). This attribute is optional.

precision: For Counter objects, the value at which the Counter will roll-over. Required for all Counter objects.

properties: A bitstring of boolean properties of the object. If the bit is on, it has the given property. This attribute is optional. The bits currently defined are:

0 -- If true, the difference between two values of this object is significant. For example, the changes in a packet count is always significant, it always conveys information. In this case, the 0 bit would be set. On the other hand, the difference between two readings of a queue length may be meaningless.

IMPLEMENTATION SPECIFIC TYPES

Each vendor or implementation specific value must be contained within an VendorSpecific object. The format of the VendorSpecific object is shown below.

Type: VendorSpecific

```
VendorSpecific ::= [APPLICATION 3] IMPLICIT SET of ANY
```

For a detailed discussion of the need for this type, see RFC 1023.

WELL-KNOWN TYPES

There are some generally useful types which are defined across the system and are considered well-known. These types support abstract notions that are frequently used in other definitions.

TYPE: Error

```
Error ::= [APPLICATION 0] IMPLICIT SEQUENCE {
    errorCode INTEGER,
    errorOffset INTEGER
    errorDescription IA5String,
}
```

The Error type is returned within reply messages when an error is countered. The errorCode is a number specifying a general class of error. The errorOffset is the octet in the query where the error was discovered. Note that the query starts at the first octet (octet 0) of the HEMP data section. The errorDescription is a text message explaining the error. Note that the definition of this section is the same (except for the start of the offset) as that of the HEMP protocol error structure and the error codes have been selected to keep the code spaces distinct. This is intended to ease the processing of error messages. The defined errorCodes are:

100 -- Any error not listed below.

101 -- System error. The query processor has failed in some way.

102 -- Format error. An error has been detected in the input stream.

103 -- Stack error. A stack overflow or underflow has occurred.

104 -- Instruction error. The instruction is either unknown, or not supported on the object to which it has been applied.

105 -- Operand error. The wrong number of operands or inappropriate operands have been given to an instruction.

TYPE: Counter

```
Counter ::= [APPLICATION 4] IMPLICIT INTEGER
```

The Counter type is an unsigned integer which is defined to roll-over to 0 when incremented past a certain value. (The roll-over point may be found by examining the attributes for the particular counter.) Counter sizes should be chosen such that the counters will not roll over more than once every 24 hours.

TYPE: InstructionGroup

```
InstructionGroup ::= [APPLICATION 5] IMPLICIT SEQUENCE
    of ANY
```

An InstructionGroup is an encapsulated sequence of operands and operations. It allows applications to encode queries as objects.

TYPE: Histogram

```
Histogram ::= SET of HistEntry

HistEntry ::= SEQUENCE {
    histValue INTEGER,
    histCount Counter
}
```

A Histogram associates a count, histCount, with a numeric value, histValue. No meaning is placed on the count or value by this definition. Each HistEntry may represent a simple map (e.g., histCount instances of histValue), or a more complex relationship (e.g., a count of all values between this histValue and the next lowest histValue in the Histogram). The meaning of the particular Histogram is given in the object definition.

TYPE: TrafficMatrix

```
TrafficMatrix ::= SET of TrafficEntry

TrafficEntry ::= SEQUENCE {
    src IpAddress,
    dst IpAddress,
    count Counter
}
```

A TrafficMatrix measures traffic observed between two IP addresses. Typically it is used to count packets flowing through a gateway.

TYPE: IpAddress

```
IpAddress ::= OCTETSTRING
```

The 4 octet IP address. If the length of the string is less than 4 then the missing octets are wildcarded. A zero length string is a default address (e.g., for indicating default routes).

TYPE: Fraction

```
Fraction ::= INTEGER
```

A Fraction is an integer representation of a fractional value. It contains the numerator of a value as expressed over 256. (For example dividing the Fraction by 256 gives the fractional value.)

TYPE: BootClock

```
BootClock ::= INTEGER
```

The time in milliseconds since the machine was last booted or reset. This value is always defined.

TYPE: localClock

```
LocalClock ::= INTEGER
```

The local system clock, measured in milliseconds since 00:00 1 January 1900 UTC. Assumed to be only a local estimate of the time. The value 0 is reserved for an uninitialized clock (For example, an uninitialized time-of-day chip.)

TYPE: NetClock

```
NetClock ::= INTEGER
```

A network synchronized clock, which is assumed to be synchronized across some part of a network. The clock value is measured in milliseconds since 00:00 1 January 1900 UTC. Specific information about the synchronization protocol is found in the system variable dictionary. The value 0 is used to indicate an uninitialized clock.

TYPE: TimeStamp

```
TimeStamp ::= CHOICE {
    [0] BootClock
```

```
            [1] localClock
            [2] NetClock
        }
```

A TimeStamp, which was taken from the boot clock, system clock or the synchronized clock. In general, a time of day is preferred to the time since boot, and a synchronized clock is preferred to an unsynchronized clock. It is more useful to know that an event occurred at a particular time, than that it happened so many milliseconds after the machine booted.

OBJECT DEFINITIONS

The Root Dictionary

In HEMS, all data is stored in dictionaries, where a dictionary is thought to represent a conceptual grouping of values. The top-level dictionary is the root dictionary. The form of the root dictionary for is shown below.

```
RootDictionary ::= [APPLICATION 32] IMPLICIT SET {
    SystemVariables,
    EventControls OPTIONAL,
    Interfaces,
    IpNetworkLayer,
    IpRoutingTable,
    IpTransportLayer,
    IpApplications OPTIONAL
}
```

The root dictionary is split into seven general dictionaries:

- SystemVariables, which stores general system values such as the system clock, machine memory and system up/down status.

- EventControls, which stores all objects necessary to observe and control the event generating mechanism in entities which support events.

- interfaces, which contains all information on all the network interfaces and IP to physical address maps (ARP tables, X.25 Standard mappings, etc).

- IpNetworkLayer, which contains information about the workings of the IP layer. This includes information such as routing tables, general packet counts, and host-traffic

matrices.

- IpRoutingTable, which contains information on how the machine routes packets. It proved more useful to segregate routing information than to keep it stored with the network layer data.

- IpTransportLayer, which stores information on the transport protocols that the entity supports.

- IpApplications, which may store information about various internet applications such as the domain system. This section is not required of HEMS entities.

The next several sections define the values stored in the five dictionaries.

The SystemVariables Dictionary

The SystemVariables dictionary stores objects which are not strictly protocol, network, or application specific. Such objects include values such as the machine load, clocks and the processor status. The form of the dictionary is shown below.

```
SystemVariables ::= [APPLICATION 33] IMPLICIT SET {
    referenceClock   [0] IMPLICIT TimeStamp,
    netClockInfo     [1] IMPLICIT SET OPTIONAL,
    processorLoad    [2] IMPLICIT INTEGER,
    entityState      [3] IMPLICIT INTEGER,
    kernelMemory     [4] IMPLICIT OCTETSTRING OPTIONAL,
    pktBuffers       [5] IMPLICIT INTEGER OPTIONAL,
    pktOctets        [6] IMPLICIT INTEGER OPTIONAL,
    pktBuffersFree   [7] IMPLICIT INTEGER OPTIONAL,
    pktOctetsFree    [8] IMPLICIT INTEGER OPTIONAL
    systemID         [9] IMPLICIT IA5STRING,
}
```

OBJECT: SystemVariables

Type: SET

Definition: see above

The objects in the dictionary are defined below.

OBJECT: referenceClock

Type: TimeStamp

Definition: The system clock used for placing timestamps on information. Use of a NetClock is encouraged.

Operations on Object: Defaults.

Notes: Cross-network clock adjustment is best handled by a proper time synchronization protocol, not through the use of SET.

OBJECT: netClockInfo

Type: SET

Definition: Detailed information on the referenceClock if the referenceClock is a NetClock. The format of this information is shown below.

```
netClockInfo ::= [1] IMPLICIT SET {
    estError INTEGER,
    refClockType INTEGER {
        unspecified(0), primary-reference(1),
        ntp-secondary-reference(2), secondary-reference(3),
        wristwatch(4)
      }
 }
```

The estError is the estimated error in milliseconds. The refClockType is a value indicating the type of reference clock consulted for network time (the values are taken directly from the Network Time Protocol specification, RFC-958).

Object Status: Required if the referenceClock is a NetClock.

OBJECT: processorLoad

Type: Fraction

Definition: A value, expressed as a Fraction, which indicates the current processing load on the entity. A value of 256 (= 1.0) is defined to be running at capacity. It is recognized that this is an imprecise definition since capacity can be measured in several ways. For example, a multiprocessor may still have plenty of capacity even if one processor is running at capacity,

or it may be at capacity because that processor is the master processor and handles all context switching. The idea is for remote applications to be able to get some sense of the current workload on the entity. Also note that the time scale of the measurement should be small. A load measure that averages over the past 10 seconds is acceptable but a load measure that averages over the past 10 minutes is not. Implementors should chose some mapping between system load and this scale such that 256 represents a machine under severe strain. (Note that this suggests that values greater than 256 may be returned in rare cases.)

OBJECT: entityState

Type: INTEGER

Definition: An object which indicates the system state. There are several defined object values. Some values are read-only and can only be read from the object. Over values are write-only and will never be read from the object. Over values are write-only and will never be read from the object.The values are:

The read-only values are:

(0) -- reserved.

(1) -- running. The entity is up and running.

(2) -- testing. The entity is running some sort of diagnostics which may affect its network operation.

The write-only values are:

(0) -- reserved.

(1) -- reset the entity.

(2) -- reboot the entity. This value is assumed to cause a more aggressive recycling of the system than reset, though this need not be the case.

(3) -- halt the entity. This value stops the entity. It assumed to prevent the entity from operating until it is manually restarted. (I.e. the halt takes the machine off the network).

Note: The ability to change an entity's state requires very strong access controls.

Operations on Object: The defaults except as noted below.

SET: Optionally writes the value into the object. The message requesting the SET must be authenticated.

SET-MATCH: Optionally writes the value into the object if the current value is matched.

OBJECT: kernelMemory

Type: OCTETSTRING

Definition: A sequence of octets which represents the image of the kernel software running on the entity. This facility is provided to allow remote network debugging.

By kernel software, we mean that software which controls the operations and access to the hardware. In particular, the kernel is expected to contain all network software up through the IP layer.

Implementations which use lightweight processes or segmented images should consider providing some way to map their internal representation into a single contiguous stream of octets.

Note: Access control is required to read this object.

Object Status: Useful.

Operations on Object: The defaults except as noted below.

GET-RANGE: Emits the section of memory specified.

GET: Emits all of memory, but note that a GET on the system dictionary should *not* emit this object.

OBJECT: pktBuffers

Type: INTEGER

Definition: The total number of packet buffers in the entity.

Object Status: Required if the entity has a maximum number of buffers. Note that most entities do have a limit (even if it

is for practical purposes, near infinite) and should return that limit.

OBJECT: pktOctets

Type: INTEGER

Definition: The maximum number of octets that can be buffered in the entity at any one time.

Object Status: Required if the entity has a maximum number of octets it can buffer. Note that most entities do have a limit and should return that limit.

OBJECT: pktBuffersFree

Type: INTEGER

Definition: The number of packet buffers currently available. Subtracting pktBuffersFree from pktBuffers should give the number of buffers in use.

Object Status: Required if there is a limit on the number of buffers.

OBJECT: pktOctetsFree

Type: INTEGER

Definition: The number of octets currently available including those not used in allocated buffers. Subtracting this value from pktOctets should give the number of octets in use.

This object can be used to track how well the entity buffers its data.

Object Status: Required if there is a limit on the number of octets that can be buffered.

OBJECT: systemID

Type: IA5STRING

Definition: The text identification of the entity. This value should include the full name of the vendor, the type of system,

and the version number of the hardware and software running on the entity.

The EventControls Dictionary

The EventControls dictionary contains objects to control and monitor the delivery of event messages to operations centers. The format of this dictionary is shown below.

```
EventControls ::= [APPLICATION 34] IMPLICIT SET OPTIONAL {
    lastEvent      [0] IMPLICIT OCTETSTRING OPTIONAL,
    eventMessageID [1] IMPLICIT Counter,
    eventCenters   [2] IMPLICIT SET of IpAddress,
    eventList      [3] IMPLICIT SET of eventEntry,
}
```

OBJECT: eventControls

Type: SET

Definition: See above.

Object Status: This object will be required in entities which support events, after the event definitions have been properly specified. See discussion of the event formats at the end of this memo.

A description of the fields in this dictionary are given below.

OBJECT: lastEvent

Type: OCTETSTRING

Definition: The last event message sent.

Object Status: Implementation of this object is encouraged if the transport protocol used for events is unreliable (e.g., UDP).

OBJECT: eventMessageID

Type: Counter

Definition: The HEMP MessageId to be used in the next event message. Equals the number of events sent.

OBJECT: eventCenters

Type: SET of IpAddress

Definition: The list of IP addresses to which events are sent. This list receives all events. For more selective event monitoring, centers should list themselves under the particular events of interest.

Note: If the SET operator is defined then use of some form of access control is recommended.

Operations on Object: The defaults except as listed below.

CREATE: Adds an address to the list. The new address may not be a broadcast address (it may be a multicast address).

DELETE: Deletes an address from the list.

SET-MATCH: Defined on the IP address. Replaces the address with a new value.

EMIT-MATCH: Defined on the IP address.

OBJECT: eventList

Type: SET of eventEntry

Definition: An array of entries which contain objects which allow management centers to control how and when events are sent. (The contents of the eventEntry structure are explained below.)

The eventControls Dictionary: eventList/eventEntry

The eventEntry provides the necessary control objects to manage how a particular event is sent. The format of the eventEntry is shown below.

```
eventEntry ::= [0] IMPLICIT SET {
    eventID        [0] IMPLICIT INTEGER,
    eventMode      [1] IMPLICIT INTEGER,
    eventCount     [2] IMPLICIT Counter,
    threshold      [3] IMPLICIT Counter,
```

```
            thresholdIncr  [4] IMPLICIT INTEGER,
            eventExecution [5] IMPLICIT InstructionGroup OPTIONAL,
            eventCenters   [6] IMPLICIT SET of IpAddress
        }
```

OBJECT: eventEntry

Type: SET

Definition: See Above.

OBJECT: eventID

Type: INTEGER

Definition: The particular event ID.

OBJECT: eventMode

Type: INTEGER

Definition: A control object which determines how and whether this event is sent. The three modes are:

0 -- unused.

1 -- off. The event is not sent.

2 -- on. The event is sent every time it occurs.

3 -- threshold. The event is sent every time the event count reaches the threshold.

OBJECT: eventCount

Type: Counter

Definition: The number of times this event has occurred.

OBJECT: threshold

Type: Counter

Definition: The event threshold. If the eventMode is "threshold" then a event is sent every time the eventCount equals this value.

Operations on Object: The defaults except as noted below.

SET: Changes the threshold.

OBJECT: thresholdIncr

Type: INTEGER

Definition: The threshold increment. Every time a event threshold is reached, the threshold value is incremented by this value (modulo the precision of the Counter) to find the new threshold.

Operations on Object: The defaults except as noted below.

SET: Changes the increment.

OBJECT: eventExecution

Type: InstructionGroup

Definition: A query to be executed whenever the event is actually sent. Any data retrieved by this query is appended to the event message.

Object Status: Encouraged.

Operations on Object: The defaults except as noted below.

SET: Changes the buffer.

OBJECT: eventCenters

Type: SET

Definition: The IP addresses of the monitoring centers which wish to listen to this particular event. Note that events should be sent to both these centers and the global list of event centers.

Operations on Object: The defaults except as noted below.

CREATE: Adds an address to the list of centers.

DELETE: Deletes an address from the list.

SET-MATCH: Defined on the IP address. Replaces the entry with a new value.

EMIT-MATCH: Defined on the IP address.

The Interfaces Dictionary

The Interfaces dictionary a list of per-interface objects. Since one of the fundamental goals of HEMS is to use generic interfaces across differing hardwares, all hardware interfaces are described by the same data structure, the InterfaceData.

```
Interfaces ::= [APPLICATION 35] IMPLICIT SET OF InterfaceData
```

OBJECT: Interfaces

Type: SET

Definition: see above.

The Interfaces Dictionary: The InterfaceData structure.

The InterfaceData structure contains all information on a particular interface. The form of the structure is shown below.

```
InterfaceData ::= [0] IMPLICIT SET {
    addresses          [0] IMPLICIT SET of IpAddress,
    mtu                [1] IMPLICIT INTEGER,
    netMask            [2] IMPLICIT IpAddress,
    pktsIn             [3] IMPLICIT Counter,
    pktsOut            [4] IMPLICIT Counter,
    inputPktsDropped   [5] IMPLICIT Counter,
    outputPktsDropped  [6] IMPLICIT Counter,
    bcastPktsIn        [7] IMPLICIT Counter OPTIONAL,
    bcastPktsOut       [8] IMPLICIT Counter OPTIONAL,
    mcastPktsIn        [9] IMPLICIT Counter OPTIONAL,
    mcastPktsOut      [10] IMPLICIT Counter OPTIONAL,
    inputErrors       [11] IMPLICIT Counter,
    outputErrors      [12] IMPLICIT Counter,
    outputQLen        [13] IMPLICIT INTEGER,
    name              [14] IMPLICIT IA5String,
    status            [15] IMPLICIT INTEGER,
    ifType            [16] IMPLICIT INTEGER,
    mediaErrors       [17] IMPLICIT Counter OPTIONAL,
```

```
            upTime          [18] IMPLICIT TimeStamp,
            broadcast       [19] IMPLICIT BITSTRING
            multicast       [20] IMPLICIT SET of BITSTRING,
            addressList     [21] IMPLICIT SET OPTIONAL,
        }
```

OBJECT: InterfaceData

Type: SET

Definition: see above.

Operations on Object: The defaults except as noted below.

SET-MATCH: This operation is optionally defined on the address field of the structure. Only certain fields in this structure may be changed. The fields which may be SET are indicated in the descriptions below.

GET-MATCH: Defined to emit information on the interface which matches the address given.

The fields in the structure are defined below.

OBJECT: addresses

Type: SET of IpAddress

Definition: The IP addresses that the interface accepts. Note that additional information on multicast addresses may be found in the IgmpValues dictionary.

OBJECT: mtu

Type: INTEGER

Definition: The maximum transmission unit of the device.

OBJECT: netMask

Type: IpAddress

Definition: The subnet mask, which is an address with all the network bits set to 1 and all the hosts bits set to 0. Used to identify subnets.

OBJECT: pktsIn

Type: Counter

Definition: The total number of packets received on this interface including those in error.

OBJECT: pktsOut

Type: Counter

Definition: The total number of packets that higher levels have attempted to send, including those that were not sent.

OBJECT: inputPktsDropped

Type: Counter

Definition: The number of good inbound packets dropped (presumably to free up buffer space).

OBJECT: outputPktsDropped

Type: Counter

Definition: The number of good outbound packets dropped (presumably to free up buffer space).

OBJECT: bcastPktsIn

Type: Counter

Definition: The number of broadcast packets received including those in error.

Object Status: Encouraged on interfaces that support broadcast.

OBJECT: bcastPktsOut

Type: Counter

Definition: The number of broadcast packets that higher levels attempted to send, including those that were not sent.

Object Status: Encouraged on interfaces that support broadcast.

OBJECT: mcastPktsIn

Type: Counter

Definition: The number of multicast packets received including those in error.

Object Status: Encouraged on interfaces that support multicast.

OBJECT: mcastPktsOut

Type: Counter

Definition: The number of multicast packets sent, including those that were not sent.

Object Status: Encouraged on interfaces that support multicast.

OBJECT: inputErrors

Type: Counter

Definition: The number of inbound packets that could not be delivered. The number of inbound packets delivered should equal inputPkts less this value and inputPktsDropped.

OBJECT: outputErrors

Type: Counter

Definition: The number of outbound packets that could not be transmitted because of errors. The number of outbound packets placed on the network should equal outputPkts less this value and outputPktsDropped.

OBJECT: outputQLen

Type: INTEGER

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

OBJECT: name

Type: IA5String

Definition: A text string completely identifying the interface. This string should include the name of the manufacturer, the product name and the version of the hardware.

OBJECT: status

Type: INTEGER

Definition: The status of the object. The status values are:

```
0 -- reserved
1 -- testing (the interface is in some test mode)
2 -- down  (the interface is down)
3 -- up  (the interface is up ready to pass packets)
```

Note: If set operations are defined, access control is required.

Operations on Object: The defaults except as noted below.

SET: Optionally defined to change the state of the interface.

OBJECT: ifType

Type: INTEGER

Definition: A flag which indicates the type of interface in use. The currently defined types are:

```
 0 -- reserved
 1 -- 1822 HDH
 2 -- 1822
 3 -- FDDI
 4 -- DDN X.25
 5 -- RFC-877 X.25
 6 -- StarLan
 7 -- Proteon 10Mbit
 8 -- Proteon 80Mbit
 9 -- Ethernet
10 -- 802.3 Ethernet
11 -- 802.4 Token Bus
12 -- 802.5 Token Ring
13 -- Point-to-Point Serial
```

OBJECT: mediaErrors

Type: Counter

Definition: A counter of media errors, such as collisions on Ethernets, token regeneration on token passing rings, or lost RFNMs on PSNs.

Object Status: Encouraged for interfaces to media which have such errors.

OBJECT: upTime

Type: TimeStamp

Definition: When the interface was put in its current state.

OBJECT: broadcast

Type: BITSTRING

Definition: Whether this interface has a physical broadcast address.

Object Status: Required if the interface has a broadcast adddress.

OBJECT: multicast

Type: SET of BITSTRING

Definition: The set of hardware multicast addresses currently enabled on the device.

Object Status: Encouraged in interfaces which support multicast.

OBJECT: addressList

Definition: SET of addressMap

```
addressMap ::= [0] IMPLICIT SET {
        ipAddr     [0] IMPLICIT IpAddress
        physAddr   [1] IMPLICIT BITSTRING
}
```

Definition: Most interfaces maintain tables mapping physical network address to IP address. An example is an ARP table. This table stores that map as a series of entries which map

IP addresses to the physical address.

Object Status: Required if the interface has to map IP addresses to physical addresses.

The IpNetworkLayer Dictionary

The IpNetworkLayer dictionary contains all information about the IP Layer. The format of the dictionary is shown below.

```
IpNetworkLayer ::= [APPLICATION 36] IMPLICIT SET {
    gateway           [0] IMPLICIT BOOLEAN,
    inputPkts         [1] IMPLICIT Counter,
    inputErrors       [2] IMPLICIT Counter,
    inputPktsDropped  [3] IMPLICIT Counter,
    inputQLen         [4] IMPLICIT INTEGER OPTIONAL,
    outputPkts        [5] IMPLICIT Counter,
    outputErrors      [6] IMPLICIT Counter,
    outputPktsDropped [7] IMPLICIT Counter,
    outputQLen        [8] IMPLICIT INTEGER OPTIONAL,
    ipID              [9] IMPLICIT Counter,
    fragCreated       [10] IMPLICIT Counter OPTIONAL,
    fragRcvd          [11] IMPLICIT Counter OPTIONAL,
    fragDropped       [12] IMPLICIT Counter OPTIONAL,
    pktsReassembled   [13] IMPLICIT Counter OPTIONAL,
    pktsFragmented    [14] IMPLICIT Counter OPTIONAL,
    htm               [15] IMPLICIT TrafficMatrix OPTIONAL,
    itm               [16] IMPLICIT TrafficMatrix OPTIONAL
}
```

OBJECT: IpNetworkLayer

Type: SET

Definition: See above.

The fields of the dictionary are defined below.

OBJECT: gateway

Type: BOOLEAN

Definition: A boolean value which is true if the entity gateways packets.

OBJECT: inputPkts

Type: Counter

Definition: The total number of input packets received including those in error.

OBJECT: inputErrors

Type: Counter

Definition: The number of input packets discarded due to errors (unknown protocols, format errors, etc).

OBJECT: inputPktsDropped

Type: Counter

Definition: The number of input packets dropped for lack of buffer space.

OBJECT: inputQLen

Type: INTEGER

Definition: The number of inbound packets currently waiting to be processed by the IP layer.

Object Status: Encouraged.

OBJECT: outputPkts

Type: Counter

Definition: The total number of outbound packets including both those packets presented to the IP layer by higher layers and packets which are gatewayed.

OBJECT: outputErrors

Type: Counter

Definition: The number of output packets discarded because of

errors (unable to route, format errors, etc).

OBJECT: outputPktsDropped

Type: Counter

Definition: The number of output packets dropped for lack of buffer space.

OBJECT: outputQLen

Type: INTEGER

Definition: The number of outbound packets waiting to be processed by the IP layer.

Object Status: Encouraged.

OBJECT: ipID

Type: Counter

Definition: The next IP packet ID identifier to be used. Note that in some implementations the transport layer may set the IP identifier, in which case this value is used if the IP identifier has not been set by the transport layer.

OBJECT: fragCreated

Type: Counter

Definition: The number of IP fragments created at this entity. (e.g., if an IP is split into three fragments at this entity, then this counter is incremented by three).

Object Status: Encouraged.

OBJECT: fragRcvd

Type: Counter

Definition: The number of IP fragments received at this entity.

Object Status: Encouraged.

OBJECT: fragDropped

Type: Counter

Definition: The number of IP fragments discarded at this entity for whatever reason (timed out, errors, etc).

Object Status: Encouraged.

OBJECT: pktsReassembled

Type: Counter

Definition: The number of IP datagrams that have been reassembled at this entity.

Object Status: Encouraged

OBJECT: pktsFragmented

Type: Counter

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

Object Status: Encouraged.

OBJECT: htm

Type: TrafficMatrix

Definition: A host traffic matrix, mapping all traffic switched any pair of sources and destinations. The count in each trafficEntry routeDst is expressed in packets. Source routed IP packets should be logged as being between their source and the destination (i.e., they should not be treated as destined for this entity).

Notes: This information may be considered sensitive.

Object Status: Encouraged in gateways.

OBJECT: itm

Type: TrafficMatrix

Definition: An interface traffic matrix showing traffic switched between interfaces in an entity. The source and destinations fields are the IP addresses of the interfaces between which the packet was switched. The count in each trafficEntry is expressed in packets.

Object Status: Useful.

The IpRoutingTable Dictionary

The IpRoutingTable dictionary contains all routing information. Note that information about any routing protocols used to maintain the routing table is found under the entry for the routing protocol. The format of the routing dictionary is shown below.

```
IpRoutingTable ::= [APPLICATION 37] IMPLICIT SET {
        routingProtocols [0] IMPLICIT OCTETSTRING,
        coreRouter       [1] IMPLICIT BOOLEAN,
        autoSys          [2] IMPLICIT INTEGER,
        metricUsed       [3] IMPLICIT OCTET,
                         [4] RoutingEntries,
}
```

OBJECT: IpRoutingTable

Type: SET

Definition: See above.

The objects contained in the dictionary are described below.

OBJECT: routingProtocols

Type: OCTETSTRING

Definition: A sparse list of the routing protocols used to update the routing table (e.g., EGP and ICMP). Each octet contains one of the following values:

0 -- anything not specified below.

1 -- local (non-protocol) information. (E.g. routing tables can be changed by hand).

2 -- HEMS (was changed/set by a HEMS operation)

3 -- Internet Control Message Protocols, (i.e. ICMP redirects).

4 -- Exterior Gateway Protocol (EGP).

5 -- Gateway-to-Gateway Protocol (GGP).

6 -- Dissimilar Gateway Protocol (DGP).

7 -- HELO

8 -- RIP

9 -- Proprietary IGP

OBJECT: coreRouter

Type: BOOLEAN

Definition: This value is set to true if this entity is a reference router for any other router (i.e., if it distributes any of its routes to other machines).

OBJECT: autoSys

Type: INTEGER

Definition: The autonomous system number of the autonomous system in which this entity resides.

OBJECT: metricUsed

Type: OCTET

Definition: Classifies the routing metric used in the routing table entries. The value should be chosen from the list of values for routingProtocols above, and indicates the metric definition used (e.g., this entity uses an EGP metric internally).

OBJECT: RoutingEntries

Type: SET of RoutingEntry

Definition: The set of all routing entries. The RoutingEntry is defined below.

The IpRoutingTable Dictionary: The RoutingEntry

The RoutingEntry contains all information on a particular route. The format of the structure is shown below.

```
RoutingEntry ::= [0] IMPLICIT SET {
    routeMetric     [0] IMPLICIT INTEGER,
    routeDst        [1] IMPLICIT IpAddress,
    nextHop         [2] IMPLICIT IpAddress,
    routeAuthor     [3] IMPLICIT IpAddress OPTIONAL,
    routeproto      [4] IMPLICIT Octet OPTIONAL,
    routeTime       [5] TimeStamp,
    routeTOS        [6] IMPLICIT INTEGER OPTIONAL,
    valid           [7] IMPLICIT BOOLEAN
}
```

OBJECT: RoutingEntry

Type: SET

Definition: See above.

Operations on Object: Defaults except as specified below.

CREATE: Adds a new routing entry. It should be confirmed that the entry is new.

DELETE: Deletes a routing entry.

GET-MATCH: The match operator is defined on the routeDst field. A match on an IpAddress is defined to be a search to find the route or routes which would be used to reach the IpAddress. More than one route may be applicable, in which case all possible routes should be returned.

SET-MATCH: Is optionally defined on the object. A SET on an entire RoutingEntry replaces the entire entry with a new value. Certain fields (indicated below) can also be changed using a SET-MATCH.

The match operator is defined on the routeDst and routeTOS fields. To SET a value, the match must be exact on the IP address (this is different from the

search definition for GET-MATCH).

Note that support of the operator on an entity which uses a dynamic routing protocol such as GGP or EGP will require close coordination with the routing protocol to ensure consistent data. (Arguably, this facility should not be supported on such machines).

The definitions of the fields in the RoutingEntry are given below.

OBJECT: routeMetric

Type: INTEGER

Definition: The routing metric on this route. Note that the type of metric is defined in the metricUsed field of the IpRoutingTable dictionary.

OBJECT: routeDst

Type: IpAddress

Definition: The final destination that can be reached via this route.

OBJECT: nextHop

Type: IpAddress

Definition: The next hop to the final destination.

OBJECT: routeAuthor

Type: IpAddress

Definition: The IP address of the entity from which this route was *first* received. That is, the first entity which stated that was reached via nextHop. The default IpAddress should be used to indicate routes which originated on the entity.

Object Status: Encouraged.

OBJECT: routeProto

Type: Octet

Definition: The routing protocol from which this route was learned. The value is taken from the list of values for routingProtocols above.

Object Status: Encouraged.

OBJECT: routeTime

Type: TimeStamp

Definition: When this route was first received.

Object Status: Encouraged.

OBJECT: routeTOS

Type: INTEGER

Definition: The IP Type of Service which this routing entry serves.

Object Status: Required if type of service routing is supported.

OBJECT: valid

Type: BOOLEAN

Definition: Whether the route is active. (Some machines retain routes which are no longer valid for various reasons.)

The IpTransportLayer Dictionary

The IpTransportLayer Dictionary contains any information which pertains to transport protocols which use the IP protocol as the network protocol. For ease of reference, the ASN.1 tag of each transport protocol's dictionary is the same as the assigned IP Protocol number. The definition of the IpTransportLayer dictionary is shown below. Note that dictionaries for many protocols are not yet defined.

```
IpTransportLayer ::= [APPLICATION 38] IMPLICIT SET {
            [0] IMPLICIT ProtocolsSupported,
            [1] IMPLICIT IcmpValues,
            [2] IMPLICIT IgmpValues OPTIONAL,
```

```
                [3] IMPLICIT GgpValues OPTIONAL,
                [7] IMPLICIT TcpValues OPTIONAL,
                [8] IMPLICIT EgpValues OPTIONAL,
                [17] IMPLICIT UdpValues OPTIONAL,
                [20] IMPLICIT HmpValues OPTIONAL,
                [27] IMPLICIT RdpValues OPTIONAL,
                [30] IMPLICIT NetbltValues OPTIONAL,
    }
```

OBJECT: IpTransportLayer

Type: SET

Definition: see above.

The objects in the dictionary are defined below.

The IpTransportLayer Dictionary: ProtocolsSupported

OBJECT: protocolsSupported

Type: OCTETSTRING

Definition: Sparse list of transport protocols supported. Each octet in the OCTETSTRING contains the IP protocol number of a supported protocol. For the purposes of this definition, an entity supports a protocol if it both contains software to makes it possible for the protocol to be used in communications with the entity, AND the entity keeps the required values (if any) defined in this memo for that protocol.

The IpTransportLayer Dictionary: IcmpValues

The IcmpValues dictionary is a subdictionary of the IpTransportLayer dictionary which tracks the workings of the Internet Control Message Protocol, defined in RFC-792. The form of the dictionary is shown below.

```
        IcmpValues ::= SET {
            inputPktCount   [0] IMPLICIT Counter,
            inputPktErrors  [1] IMPLICIT Counter,
            inputPktDeliver [2] IMPLICIT Counter,
            inputPktTypes   [3] IMPLICIT Histogram OPTIONAL,
            outputPktCount  [4] IMPLICIT Counter,
            outputPktErrors [5] IMPLICIT Counter,
            outputPktTypes  [6] IMPLICIT Histogram OPTIONAL,
            icmpTraffic     [7] IMPLICIT TrafficMatrix OPTIONAL,
            ipID            [8] IMPLICIT Counter OPTIONAL
```

```
        }
```

OBJECT: IcmpValues

Type: SET

Definition: see above.

The objects in the dictionary are defined below.

OBJECT: inputPktCount

Type: Counter

Definition: The total number of ICMP packets received (including those in error).

OBJECT: inputPktErrors

Type: Counter

Definition: The number of ICMP packets received which proved to have errors (bad checksums, bad length etc). Subtracting this value from the inputPktCount field should give the number of valid ICMP packets received.

OBJECT: inputPktDeliver

Type: Counter

Definition: The number of valid ICMP packets which were successfully processed (e.g., delivered to the higher protocol).

OBJECT: inputPktTypes

Type: Histogram

Definition: A histogram of ICMP messages types and codes received, not including those messages that proved to contain errors. The histogram histValue field contains a 16-bit value which is the the (ICMP type * 256) + ICMP code, and the histCount field contains the number of valid messages containing this

type/code pair which have been received.

The message type and code values are those defined in RFC-792 (e.g., the Time Exceeded Message with a code of "fragment reassembly time exceeded" is (11 * 256) + 1 = 2817).

Object Status: Useful.

Operations on Object: The defaults except as listed below:

GET-MATCH: Match is defined on the value of the histValue field.

OBJECT: outputPktCount

Type: Counter

Definition: The total number of ICMP packets that the entity attempted to send (including those that failed due to lack of buffers, a missing route or other transient transmission problems).

OBJECT: outputPktErrors

Type: Counter

Definition: The number of ICMP packets which the entity could not send due to transmission problems such as the lack of buffers, a missing route or other transient transmission problems. This value is not required to include errors which the ICMP layer could not reasonably be expected to detect such as damage to the packet in transit. Subtracting this value from the PktCount field should give the number of ICMP packets the entity believes it successfully sent.

OBJECT: outputPktTypes

Type: Histogram

Definition: A histogram of ICMP messages types and codes sent, including those messages that later failed to be transmitted. The histogram histValue field contains a 16-bit value which is the the (ICMP type * 256) + ICMP code, and the histCount field contains the number of valid messages containing this type/code pair which have been sent.

The message type and code values are those defined in RFC-792 (e.g., the Time Exceeded Message with a code of "fragment reassembly time exceeded" is (11 * 256) + 1 = 2817).

Object Status: Useful.

Operations on Object: The defaults except as listed below:

GET-MATCH: Match is defined on the value of the histValue field.

OBJECT: icmpTraffic

Type: TrafficMatrix

Definition: All ICMP traffic which has originated on this machine. The source address in the traffic matrix should be the interface from which the packet was sent. The destination is the address to which the packet is to finally be delivered (not an intermediate hop).

Object Status: Useful.

OBJECT: ipID

Type: Counter

Definition: The next IP packet ID identifier to be used by the ICMP code.

Object Status: Required if the ICMP code generates its own IP identifiers.

The IpTransportLayer Dictionary: IgmpValues

```
IgmpValues ::= SET {
    conformance     [0] IMPLICIT INTEGER,
    inputPktCount   [1] IMPLICIT Counter,
    inputPktErrors  [2] IMPLICIT Counter,
    inputPktTypes   [3] IMPLICIT Histogram OPTIONAL,
    outputPktCount  [4] IMPLICIT Counter,
    outputPktErrors [5] IMPLICIT Counter,
    outputPktTypes  [6] IMPLICIT Histogram OPTIONAL,
    igmpTraffic     [7] IMPLICIT TrafficMatrix OPTIONAL
```

```
    igmpGroups      [8] IMPLICIT SET of IgmpGroupEntry,
    ipID            [9] IMPLICIT Counter OPTIONAL,
   }
```

OBJECT: IgmpValues

Type: SET

Definition: The dictionary of information on the Internet Group Management Protocol (RFC-988).

Object Status: Required in hosts which support IGMP.

The objects stored in this dictionary are defined below.

OBJECT: conformance

Type: INTEGER

Definition: The level of conformance with RFC-988. The conformance levels are:

0 -- Level 0. No support for IP multicasting

1 -- Level 1. Support for sending but not receiving multicast datagrams.

2 -- Level 2. Full support for IP multicasting.

These values are taken directly from RFC-988.

OBJECT: inputPktCount

Type: Counter

Definition: The number of IGMP packets received including those that proved to be in error.

OBJECT: inputPktErrors

Type: Counter

Definition: The number of IGMP packets received which proved to be in error. This value subtracted from inputPktCount should give the number of valid IGMP packets received.

OBJECT: inputPktTypes

Type: Histogram

Definition: A histogram of IGMP messages types and codes sent, including those messages that later failed to be transmitted. The histogram histValue field contains a 16-bit value which is the the (IGMP type * 256) + IGMP code, and the histCount field contains the number of valid messages containing this type/code pair which have been sent.

The type and code values are taken from RFC-988.

OBJECT: outputPktCount

Type: Counter

Definition: The total number of IGMP packets that the entity attempted to send (including those that failed due to lack of buffers, a missing route or other transient transmission problems).

OBJECT: outputPktErrors

Type: Counter

Definition: The number of IGMP packets which the entity could not send due to transmission problems such as the lack of buffers, a missing route or other transient transmission problems. This value is not required to include errors which the IGMP layer could not reasonably be expected to detect such as damage to the packet in transit. Subtracting this value from the outputPktCount field should give the number of IGMP packets the entity believes it successfully sent.

OBJECT: outputPktTypes

Type: Histogram

Definition: A histogram of IGMP messages types and codes sent, including those messages that later failed to be transmitted. The histogram histValue field contains a 16-bit value which is the the (IGMP type * 256) + IGMP code, and the histCount field contains the number of valid messages containing this type/code pair which have been sent.

The type and code values are taken from RFC-988.

OBJECT: igmpTraffic

Type: TrafficMatrix

Definition: All IGMP traffic which has originated on this machine. The source address in the traffic matrix should be the interface from which the packet was sent. The destination is the address to which the packet is to finally be delivered (not an intermediate hop).

Object Status: Useful.

OBJECT: igmpGroups

Type: SET

Definition: The various igmpGroups of which this host is aware. This is stored as a set of IgmpGroupEntry. The format of an IgmpGroupEntry is shown below.

```
IgmpGroupEntry ::= [0] SET {
    groupAddress      [0] IMPLICIT IpAddress,
    groupAccessKey    [1] IMPLICIT OCTETSTRING,
    groupAgent        [2] IMPLICIT BOOLEAN,
}
```

The groupAddress is the multicast IP address. The groupAccessKey is the 8 octet key -- this key may be confidential and should only be available to authorized querying entities. The groupAgent field is true if this entity is an agent for the multicast group.

OBJECT: ipID

Type: Counter

Definition: The next IP packet ID identifier to be used by the IGMP code.

Object Status: Required if the IGMP code generates its own IP identifiers.

The IpTransportLayer Dictionary: GgpValues

The definition of the GgpValues dictionary is left for further study.

The IpTransportLayer Dictionary: TcpValues

The TcpValues dictionary is a subdictionary of the IpTransportLayer dictionary which tracks the workings of the Transmission Control Protocol, defined in RFC-793. The definitions of several objects in this dictionary refer to definitions in RFC-793. The form of the dictionary is shown below.

```
TcpValues ::= SET {
                [0] IMPLICIT TcpParam
                [1] IMPLICIT TcpStats OPTIONAL,
  tcpConnData   [2] IMPLICIT SET of TcpConn,
}
```

OBJECT: TcpValues

Type: IMPLICIT SET

Definition: see above.

Object Status: Required if the entity supports TCP.

The objects in the dictionary are defined in the next few sections.

The IpTransportLayer Dictionary: TcpValues/TcpParam

The TcpParam dictionary contains information about certain parameters such as round-trip timer estimation constants which are managed on a per-machine basis. The form of the dictionary is shown below.

```
TcpParam ::= SET {
    tcpRtoA       [0] IMPLICIT IA5String,
    tcpRtoParam   [1] IMPLICIT SET of RtoParam,
    ipID          [2] IMPLICIT Counter,
    tcpRtoMin     [3] IMPLICIT INTEGER OPTIONAL,
    tcpRtoMax     [4] IMPLICIT INTEGER OPTIONAL,
    tcpMaxSegSiz  [5] IMPLICIT INTEGER,
    tcpMaxConn    [6] IMPLICIT INTEGER OPTIONAL,
    tcpMaxWindow  [7] IMPLICIT INTEGER OPTIONAL,
}
```

OBJECT: tcpParam

Type: SET

Definition: see above.

The definition of the objects in the tcpParam dictionary are given below.

OBJECT: tcpRtoA

Type: IA5String

Definition: The TCP retransmission timeout algorithm used. The algorithm is expressed as one or more equations to generate a target value, "RTO[N]", which is the retransmission timeout for packet "N". Expressions should use well understood symbols such as * for multiplication and / for division, and parentheses to indicate precedence. Variables should begin with an upper case character. Multiple equations should be separated by semi-colons. Comments should be in braces (i.e., {}). Constants should begin with a lower case character. In addition to "RTO[N]" the symbol "S[N]" is defined to mean the round-trip sample for packet N. Using this syntax, the algorithm in RFC-793 would be expressed as:

```
RTO[N] = SRTT[N] * beta ;
SRTT[N] = ( S[N-1] * alpha) + (SRTT[N-1] * (1 - alpha))
```

Note: The syntax probably needs to be refined so that it can be parsed and interpreted by a program. This is left for future study.

OBJECT: tcpRtoParam

Type: SET of RtoParam

Definition: The list of the values of the constants used by the retransmission timeout algorithm. The format of the RtoParam structure is shown below.

```
RtoParam ::= SEQUENCE {
    name IA5String,
    value Fraction
}
```

The name is the name of the constant as expressed in the tcpRtoA (e.g., "beta").

OBJECT: ipID

Type: Counter

Definition: The next IP packet ID identifier to be used by the TCP code.

Object Status: Required if the TCP code generates its own IP identifiers.

OBJECT: tcpRtoMin

Type: INTEGER

Definition: The minimum value the TCP implementation permits for the retransmission timeout (RTO), measured in milliseconds.

Note: If the SET operation is optionally defined, access control must be exercised.

Object Status: Required if the implementation uses the suggested algorithm in RFC-793 or if the implementation sets any limits on the minimum RTO.

Operations on Object: The defaults except as listed below:

SET: Optionally defined to change the value. Implementations should confirm that the new value is less than tcpRtoMax.

OBJECT: tcpRtoMax

Type: INTEGER

Definition: The maximum value the TCP implementation permits for the retransmission timeout (RTO), measured in milliseconds.

Note: If the SET operation is optionally defined, access control must be exercised.

Object Status: Required if the implementation uses the suggested algorithm in RFC-793 or if the implementation sets any limits on the maximum RTO.

Operations on Object: The defaults except as listed below:

SET: Optionally defined to change the value. Implementations should confirm that the new value is greater than tcpRtoMax, and that the value is large (i.e., several seconds).

OBJECT: tcpMaxSegSiz

Type: INTEGER

Definition: The maximum segment size used by this implementation.

Object Status: Required if the entity sets an upper limit on the MTU. (Some implementations have no constraints, but chose an MTU from external constraints such as the maximum MTU of the network interface in use.)

OBJECT: tcpMaxConn

Type: INTEGER

Definition: An optional value, which must be present if the entity has a limit on the total number of TCP connections it can support.

Object Status: Required if the entity sets limits.

Note: If the SET operation is defined, access control must be exercised.

Operations on Object: The defaults except as listed below:

SET: Optionally defined to change the value. If the new value is less than the number of currently open connections, implementations are *not* required to close existing connections, but may not open any additional ones.

OBJECT: tcpMaxWindow

Type: INTEGER

Definition: An optional value, which must be present if the entity places a fixed upper limit on the size of any connection's TCP window (i.e., if the maximum window size is not per connection configurable).

Object Status: Required if the entity sets limits.

Note: If the SET operation is defined, access control must be exercised.

Operations on Object: The defaults except as listed below:

SET: Optionally defined to change the value. The new value must be at least the size of one maximum TCP segment.

The IpTransportLayer Dictionary: TcpValues/TcpStats

The TcpStats dictionary stores general information about the workings of the TCP layer. The form of the dictionary is shown below.

```
TcpStats ::= SET {
     connAttempts     [0] IMPLICIT Counter OPTIONAL,
     connOpened       [1] IMPLICIT Counter OPTIONAL,
     connAccepted     [2] IMPLICIT Counter OPTIONAL,
     connClosed       [3] IMPLICIT Counter OPTIONAL,
     connAborted      [4] IMPLICIT Counter OPTIONAL,
     connAbortedInfo  [5] IMPLICIT Histogram OPTIONAL,
     octetsIn         [6] IMPLICIT Counter OPTIONAL,
     octetsOut        [7] IMPLICIT Counter OPTIONAL,
     octetsInDup      [8] IMPLICIT Counter OPTIONAL,
     octetsRetrans    [9] IMPLICIT Counter OPTIONAL,
     inputPkts       [10] IMPLICIT Counter OPTIONAL,
     retransPkts     [11] IMPLICIT Counter OPTIONAL,
     outputPkts      [12] IMPLICIT Counter OPTIONAL,
     dupPkts         [13] IMPLICIT Counter OPTIONAL,

}
```

OBJECT: TcpStats

Type: SET

Definition: See above.

Object Status: Encouraged.

The definition of the fields in the dictionary are given below.

OBJECT: connAttempts

Type: Counter

Definition: The number of connection attempts that have been made from this host. This includes pending attempts.

Object Status: Encouraged.

OBJECT: connOpened

Type: Counter

Definition: The number of connection attempts from this host which successfully generated an open connection. This includes currently open connections.

Object Status: Encouraged.

OBJECT: connAccepted

Type: Counter

Definition: The number of connections accepted by listening peers on this entity. This includes currently open connections.

Object Status: Encouraged.

OBJECT: connClosed

Type: Counter

Definition: The number of connections which were properly closed.

Object Status: Encouraged.

OBJECT: connAborted

Type: Counter

Definition: The number of connections which were aborted. Note that if implementations trace how the connection was aborted, they are encouraged to use the connAbortedInfo histogram.

Object Status: Encouraged.

OBJECT: connAbortedInfo

Type: Histogram

Definition: The number of connections which were aborted by type of abort. The histValue is one of the codes listed below. The histCount is the number of connections aborted for this reason. The histValues codes are:

```
0 -- an abort condition not specified below
1 -- remote abort
2 -- local application abort
3 -- local protocol level abort
```

Object Status: Useful

OBJECT: octetsIn

Type: Counter

Definition: The total number of TCP octets (not including duplicates) received at this entity.

Object Status: Required if TcpStats is implemented.

OBJECT: octetsOut

Type: Counter

Definition: The total number of TCP octets (not including retransmissions) sent from this entity.

Object Status: Required if TcpStats is implemented.

OBJECT: octetsInDup

Type: Counter

Definition: The total number of TCP octets received which were duplicates.

Object Status: Required if TcpStats is implemented.

OBJECT: octetsReTrans

Type: Counter

Definition: The total number of TCP octets which have been retransmitted.

Object Status: Required if TcpStats is implemented.

OBJECT: inputPkts

Type: Counter

Definition: The total number of valid packets received, including those on current connections.

Object Status: Useful.

OBJECT: retransPkts

Type: Counter

Definition: The total number of packets retransmitted.

Object Status: Useful.

OBJECT: outputPkts

Type: Counter

Definition: The total number of packets sent.

Object Status: Useful.

OBJECT: dupPkts

Type: Counter

Definition: The number of packets received which contained only data already received.

Object Status: Useful.

The IpTransportLayer Dictionary: TcpValues/TcpConn

The tcpConnData field in the TcpValues dictionary is a set of TcpConn, where each TcpConn contains information on a particular TCP connection. The definition of TcpConn is shown below.

```
TcpConn ::= SET {
    localPort       [0] IMPLICIT INTEGER,
    localAddress    [1] IMPLICIT IpAddress,
    foreignPort     [2] IMPLICIT INTEGER,
    foreignAddress  [3] IMPLICIT IpAddress,
    state           [4] IMPLICIT INTEGER,
    snduna          [5] IMPLICIT INTEGER,
    sndnxt          [6] IMPLICIT INTEGER,
    sndwnd          [7] IMPLICIT INTEGER,
    congwnd         [8] IMPLICIT INTEGER,
    rcvnxt          [9] IMPLICIT INTEGER,
    rcvwnd         [10] IMPLICIT INTEGER,
    srtt           [11] IMPLICIT INTEGER OPTIONAL,
    lastrtt        [12] IMPLICIT INTEGER OPTIONAL,
    maxSegSize     [13] IMPLICIT INTEGER,
    octetsSent     [14] IMPLICIT Counter OPTIONAL,
    octetsRXmit    [15] IMPLICIT Counter OPTIONAL,
    octetsRcvd     [16] IMPLICIT Counter OPTIONAL,
    octetDups      [17] IMPLICIT Counter OPTIONAL,
    octetPastWin   [18] IMPLICIT Counter OPTIONAL,
    segSizes       [19] IMPLICIT Histogram OPTIONAL,
}
```

The set of TCP connections can be searched in a number of ways based on the local and foreign addresses (including the port number). Individual values of a connection cannot be retrieved without a search.

OBJECT: TcpConn

Type: SET

Definition: The per TCP connection data.

Operations on Object: The defaults except as listed below:

GET-MATCH: Defined on any combination of values of localAddress, localPort, foreignAddress and foreignPort. Returns all connections which match the template. (For example, GET-MATCH on a particular foreignAddress returns all connections to that address.)

The definitions of the fields of the tcpConn structure are given below.

OBJECT: localPort

Type: INTEGER

Definition: The local port number of this connection.

Operations on Object: Defaults. Note that MATCH operators may be applied to this object to locate information on a particular TCP connection.

OBJECT: localAddress

Type: IpAddress

Definition: The local IP address of this connection. May be the default IP address defined above. This value may not be valid in certain states.

Operations on Object: Defaults. Note that MATCH operators may be applied to this object to locate information on a particular TCP connection.

OBJECT: foreignPort

Type: INTEGER

Definition: The foreign port number of this connection. This value may be meaningless if the local peer is in certain states (e.g., LISTEN).

Operations on Object: Defaults. Note that MATCH operators may be applied to this object to locate information on a particular TCP connection.

OBJECT: foreignAddress

Type: IpAddress

Definition: The foreign IP address of this connection. This value may be meaningless if the local peer is in certain states (e.g., LISTEN).

Operations on Object: Defaults. Note that MATCH operators may be

applied to this object to locate information on a particular TCP connection.

OBJECT: state

Type: INTEGER

Definition: The current state of the local peer. The values corresponding to the different states are: close(0), listen(1), syn-sent(2), syn-received(3), established(4), close-wait(5), fin-wait-1(6), closing(7), last-ack(8), fin-wait-2(9), time-wait(10). Implementations must map internal representations of the state into these values.

OBJECT: snduna

Type: INTEGER

Definition: The SND.UNA value as defined in RFC-793.

OBJECT: sndnxt

Type: INTEGER

Definition: The SND.NXT value as defined in RFC-793.

OBJECT: sndwnd

Type: INTEGER

Definition: The SND.WND value as defined in RFC-793.

OBJECT: congwnd

Type: INTEGER

Definition: The congestion window. This value is less than or equal to sndwnd. If less than sndwnd, then congestion control is in effect and congwnd is the reduced send window size in use.

OBJECT: rcvnxt

Type: INTEGER

Definition: The RCV.NXT value as defined in RFC-793.

OBJECT: rcvwnd

Type: INTEGER

Definition: The RCV.WND value as defined in RFC-793.

OBJECT: srtt

Type: INTEGER

Definition: The smoothed round-trip time in milliseconds.

Object Status: Required if the implementation maintains a smoothed round-trip time.

OBJECT: lastrtt

Type: INTEGER

Definition: The last round-trip time sample taken in milliseconds.

Object Status: Encouraged.

OBJECT: maxSegSize

Type: INTEGER

Definition: The maximum segment size that can be used on this connection.

OBJECT: octetsSent

Type: Counter

Definition: The total number of octets transmitted since the connection was opened, not including retransmissions. Can alternatively be thought of as the current length of the stream.

Object Status: Encouraged.

OBJECT: octetsRXmit

Type: Counter

Definition: The total number of octets retransmitted since the connection was opened. This plus octetsSent should give the total number of octets sent.

Object Status: Encouraged.

OBJECT: octetsRcvd

Type: Counter

Definition: The number of octets received since the connection was opened, not including duplicates received. The receiver's version of octetsSent.

Object Status: Encouraged.

OBJECT: octetDups

Type: Counter

Definition: The total number of octets received since the connection was opened which were redundant (i.e., they had been previously received).

Object Status: Encouraged.

OBJECT: octetPastWin

Type: Counter

Definition: The number of segments which contained data beyond the upper edge of the receive window.

Object Status: Encouraged

OBJECT: segSizes

Type: Histogram

Definition: A histogram of the sizes of the packets sent on the

connection (useful for catching cases of silly-window syndrome). This histogram is an range histogram, measuring the number of segments whose size fell into a give range. The histogram histValue field contains a segment size, and the histCount field contains the number of segments between this size and the next largest size.

Object Status: Useful.

The IpTransportLayer Dictionary: EgpValues

The EgpValues dictionary stores information about the workings of the Exterior Gateway Protocol, defined in RFC-904. The format of the dictionary is shown below.

```
EgpValues ::= SET {
    egpState  [0] IMPLICIT INTEGER,
              [1] IMPLICIT EgpParam,
              [2] IMPLICIT EgpStats OPTIONAL,
  egpPeerData [3] IMPLICIT SET of EgpPeer
}
```

OBJECT: EgpValues

Type: SET

Definition: See above.

Object Status: Required in entities which support EGP.

The definitions of the subdictionaries of this dictionary are given below.

OBJECT: egpState

Type: INTEGER

Definition: The state of the EGP system. The state values are:

```
0 -- Idle
1 -- Acquisition
2 -- Down
3 -- Up
4 -- Cease
```

These values are taken directly from RFC-904.

The IpTransportLayer Dictionary: EgpValues/EgpParam

The EgpParam dictionary stores the various EGP parameters. The format of the dictionary is shown below.

```
EgpParam ::= SET {
    p1    [0] IMPLICIT INTEGER,
    p2    [1] IMPLICIT INTEGER,
    p3    [2] IMPLICIT INTEGER,
    p4    [3] IMPLICIT INTEGER,
    p5    [4] IMPLICIT INTEGER,
    ipID  [5] IMPLICIT Counter OPTIONAL
}
```

OBJECT: EgpParam

Type: SET

Definition: See above.

The definition of the fields of the dictionary are given below. All the definitions are taken from RFC-904.

OBJECT: p1

Type: INTEGER

Definition: Minimum interval acceptable between successive Hello commands received.

Operations on Object: The defaults except as noted below.

SET: The set command is optionally defined on this object.

OBJECT: p2

Type: INTEGER

Definition: Minimum interval acceptable between successive Poll commands received.

Operations on Object: The defaults except as noted below.

SET: The set command is optionally defined on this object.

OBJECT: p3

Type: INTEGER

Definition: Interval between Request or Cease command retransmissions.

Operations on Object: The defaults except as noted below.

SET: The set command is optionally defined on this object.

OBJECT: p4

Type: INTEGER

Definition: Interval during which state variables are maintained in the absence of commands or response in the Down and Up states.

Operations on Object: The defaults except as noted below.

SET: The set command is optionally defined on this object.

OBJECT: p5

Type: INTEGER

Definition: Interval during which state variables are maintained in the absence of commands or response in the Acquisition and Cease states.

Operations on Object: The defaults except as noted below.

SET: The set command is optionally defined on this object.

OBJECT: ipID

Type: Counter

Definition: The next IP packet ID identifier to be used by the EGP code.

Object Status: Required if the EGP code generates its own IP identifiers.

The IpTransportLayer Dictionary: EgpValues/EgpStats

The EgpStats dictionary keeps statistics about the use of EGP on this entity. The form of the dictionary is shown below.

```
EgpStats ::= SET {
    inputPktCount   [1] IMPLICIT Counter,
    inputPktErrors  [2] IMPLICIT Counter,
    inputPktTypes   [3] IMPLICIT Histogram OPTIONAL,
    outputPktCount  [4] IMPLICIT Counter,
    outputPktErrors [5] IMPLICIT Counter,
    outputPktTypes  [6] IMPLICIT Histogram OPTIONAL,
    egpTraffic      [7] IMPLICIT TrafficMatrix OPTIONAL
}
```

OBJECT: EgpStats

Type: SET

Definition: See above.

The definitions of the objects in this dictionary are given below.

OBJECT: inputPktCount

Type: Counter

Definition: The number of EGP packets received including those that proved to be in error.

OBJECT: inputPktErrors

Type: Counter

Definition: The number of EGP packets received which proved to be in error. This value subtracted from inputPktCount should give the number of valid EGP packets received.

OBJECT: inputPktTypes

Type: Histogram

Definition: A histogram of types of valid EGP messages received. The histogram histValue field contains the message type number, and the histCount field contains the number of messages of

this type which have been received.

Object Status: Useful.

OBJECT: outputPktCount

Type: Counter

Definition: The total number of EGP packets that the entity attempted to send (including those that failed due to lack of buffers, a missing route or other transient transmission problems).

OBJECT: outputPktErrors

Type: Counter

Definition: The number of EGP packets which the entity could not send due to transmission problems such as the lack of buffers, a missing route or other transient transmission problems. This value is not required to include errors which the EGP layer could not reasonably be expected to detect such as damage to the packet in transit. Subtracting this value from the outputPktCount field should give the number of EGP packets the entity believes it successfully sent.

OBJECT: outputPktTypes

Type: Histogram

Definition: A histogram of EGP messages types sent, including those that later failed to be transmitted. The histogram histValue field contains the message type number, and the histCount field contains the number of messages of this type which have been sent.

Object Status: Useful.

OBJECT: egpTraffic

Type: TrafficMatrix

Definition: All EGP traffic which has originated on this machine. The source address in the traffic matrix should be the interface from which the packet was sent. The destination is the address

to which the packet is to finally be delivered (not an intermediate hop).

Object Status: Useful.

The IpTransportLayer Dictionary: EgpValues/EgpPeer

The egpPeerData field of the EgpValues dictionary is a set of EgpPeer structures which contain the state variables for a particular EGP neighbor. The form of the EgpPeer structure is shown below.

```
EgpPeer ::= SET {
    r       [0] IMPLICIT Counter,
    s       [1] IMPLICIT Counter,
    t1      [2] IMPLICIT INTEGER,
    t2      [3] IMPLICIT INTEGER,
    t3      [4] IMPLICIT INTEGER,
    m       [5] IMPLICIT BOOLEAN,
    timer1  [6] IMPLICIT INTEGER,
    timer2  [7] IMPLICIT INTEGER,
    timer3  [8] IMPLICIT INTEGER,
    addr    [9] IMPLICIT IpAddress
}
```

OBJECT: EgpPeer

Type: SET

Definition: The state information for a given EGP neighbor.

The definition of each field is given below.

OBJECT: r

Type: Counter

Definition: The receive sequence number as defined in RFC-904.

OBJECT: s

Type: Counter

Definition: The send sequence number as defined in RFC-904.

OBJECT: t1

Type: INTEGER

Definition: The interval between Hello command retransmissions as defined in RFC-904.

OBJECT: t2

Type: INTEGER

Definition: The interval between Poll command retransmissions as defined in RFC-904.

OBJECT: t3

Type: INTEGER

Definition: The interval during which neighbor-reachability indications are counted, as defined in RFC-904.

OBJECT: m

Type: BOOLEAN

Definition: The Hello Polling mode. True if in active mode, false if in passive mode.

Operations on Object: The defaults except as noted below.

SET: Optionally defined to change the Hello Polling mode.

OBJECT: timer1

Type: INTEGER

Definition: The value of timer 1 as defined in RFC-904.

OBJECT: timer2

Type: INTEGER

Definition: The value of timer 2 as defined in RFC-904.

OBJECT: timer3

Type: INTEGER

Definition: The value of timer 3 as defined in RFC-904.

OBJECT: addr

Type: IpAddress

Definition: The IP address of the neighbor.

The IpTransportLayer Dictionary: UdpValues

The UdpValues dictionary stores all information on the User Datagram Protocol, defined in RFC-768. The format of the dictionary is shown below.

```
UdpValues ::= [17] IMPLICIT SET OPTIONAL {
    ipID        [0] IMPLICIT Counter OPTIONAL,
                [1] IMPLICIT UdpStats,
    udpPortData [2] IMPLICIT SET of udpPort
}
```

OBJECT: UdpValues

Type: SET

Definition: See above.

Object Status: Implementation of this dictionary is required if the entity supports UDP.

The fields of this dictionary are given below.

OBJECT: ipID

Type: Counter

Definition: The next IP packet ID identifier to be used by the UDP code.

Object Status: Required if the UDP code generates its own IP identifiers.

The IpTransportLayer Dictionary: UdpValues/UdpStats

The UdpStats dictionary stores general information about the

behavior of the UDP protocol on the entity. The format of the dictionary is shown below.

```
UdpStats ::= SET {
    inputPkts       [0] IMPLICIT Counter,
    inputPktErrors  [1] IMPLICIT Counter,
    outputPkts      [2] IMPLICIT Counter,
}
```

OBJECT: UdpStats

Type: SET

Definition: See above.

Object Status: Encouraged.

The fields in this dictionary are defined below.

OBJECT: inputPkts

Type: Counter

Definition: The total number of UDP packets received at this entity including any errors.

Object Status: Required if the UdpStats dictionary is implemented.

OBJECT: inputPktsErrors

Type: Counter

Definition: The number of UDP packets which could not be delivered because of format errors, data corruption or because there was no application at the destination port.

Object Status: Required if the UdpStats dictionary is implemented.

OBJECT: outputPkts

Type: Counter

Definition: The total number of UDP segments sent from this entity.

Object Status: Required if the UdpStats dictionary is implemented.

The IpTransportLayer Dictionary: UdpValues/udpPortData

The udpPortData structure stores information about individual UDP applications. The udpPortData is represented as a set of records, udpPorts, which track the behavior of individual ports. The format of both structures are shown below.

```
udpPortData    [1] IMPLICIT SET of UdpPort

UdpPort ::=  [0] IMPLICIT SET {
    localAddress     [0] IMPLICIT IpAddress,
    localPort        [1] IMPLICIT INTEGER,
    foreignAddress   [2] IMPLICIT IpAddress OPTIONAL,
    foreignPort      [3] IMPLICIT INTEGER OPTIONAL,
    maxPktSize       [4] IMPLICIT INTEGER,
    pktsRcvd         [5] IMPLICIT Counter,
    octetRcvd        [6] IMPLICIT Counter OPTIONAL,
    pktsSent         [7] IMPLICIT Counter,
    octetSent        [8] IMPLICIT Counter OPTIONAL,
}
```

OBJECT: udpPortData

Type: SET of udpPort

Definition: See above.

OBJECT: UdpPort

Type: SET

Definition: See above.

Operations on Object: The defaults except as noted below.

GET-MATCH. Defined on any combination of the values of localAddress, localPort, foreignAddress and foreignPort. Returns all ports which match the template.

The meaning of the individual fields of the udpPort record are given below.

OBJECT: localAddress

Type: IpAddress

Definition: The local IP address of the port. May be the default IP address if records are accepted from any interface.

OBJECT: localPort

Type: INTEGER

Definition: The local port number.

OBJECT: foreignAddress

Type: IpAddress

Definition: Some UDP implementations permit applications to specify the remote address from which packets will be accepted. In such implementations, this field may be used to return the remote IP address. If this value is set to the default IP address, then packets from any host are accepted. The default IP address indicates that the application has not specified the remote address (but could if it chose).

Object Status: Required in entities which permit applications to specify the remote address.

OBJECT: foreignPort

Type: INTEGER

Definition: Some UDP implementations permit applications to specify the remote address from which packets will be accepted. In such implementations, this field may be used to return the remote port. If this value is set to 0, packets from any remote port are accepted.

Object Status: Required in entities which permit applications to specify the remote port.

OBJECT: maxPktSize

Type: INTEGER

Definition: The maximum UDP packet size, if any, supported by this host.

Object Status: Required if there is a limit on the UDP packet size.

OBJECT: pktsRcvd

Type: Counter

Definition: The total number of packets received on this port during the lifetime of this application (i.e., application which opened this port).

OBJECT: octetsRcvd

Type: Counter

Definition: The total number of octets received at this port.

OBJECT: pktsSent

Type: Counter

Definition: The total number of packets sent on this port during the lifetime of this application (i.e., the application which opened this port).

OBJECT: octetsSent

Type: Counter

Definition: The total number of octets sent on this port during the lifetime of this application (i.e., the application which opened this port).

The IpTransportLayer Dictionary: HmpValues

The HmpValues dictionary stores all information on the Host Monitoring Protocol, defined in RFC-869. Since HEMS is designed to replace HMP, the definition of this dictionary has been deferred until a clear need for it is demonstrated.

The IpTransportLayer Dictionary: RdpValues

The RdpValues dictionary stores all information on the Reliable Data Protocol (RDP). Since RDP is currently being tested and revised, the definition of this dictionary is left for further study.

The IpTransportLayer Dictionary: NetbltValues

The NetbltValues dictionary stores all information on the Network Block Transfer protocol. Since Netblt is currently being tested and revised, the definition of this dictionary is left for further study.

The IpApplications Dictionary

The IpApplications dictionary stores information about networking applications whose operations may affect the proper operation of the network. Examples of such applications might be domain nameservers or distributed routing agents (such as gated or routed). The definition of this dictionary is left for further study.

NOTES ON RETRIEVAL OF OBJECTS

It is assumed in this system that the query processor is only one of many concurrently running processes on an entity, and that the operations of the other processes may affect the values of the objects managed by the query processor. To permit this concurrency, the query processor is not required to keep the values frozen during the execution of a query. As a result, related values may change during the course of the query's execution. Applications should be prepared for this possibility.

In several places, specific mathematical relations between objects have been specified, for example, that object X minus object Y should yield some well-defined value. Note that in many cases, objects X and Y are roll-over counters, in which case these relations are only valid modulo the precision of the counter. This is acceptable. The relationships are only intended to clarify the association between objects.

EVENTS

In the remainder of this memo we present the format and definition of event messages which are unsolicited updates sent from entities to management centers.

This section needs much further work. The authors provide this section to illustrate how the trap mechanism works. However, much more research must be done into the questions of what events need to be reported, and what information they must carry with them

before this section can be completed. The authors welcome any advice from the community on this subject.

Format of Event Messages

Event messages have the same format as replies; they are a sequence of objects. The only difference between a event message and a regular reply to a query is that the event message is labelled as a event in the HEMP message header and the first object in the event message is a special event leader describing the event. All objects after the event message are standard objects stored by the entity which might be useful to a monitoring center in understanding the machine state which caused the event. Each event has a certain number of objects that it must return. Additional objects may be returned by loading instructions into the eventExecution buffer of the relevant eventEntry.

The format of the event leader is shown below:

```
EventLeader ::= [APPLICATION 1024] IMPLICIT SEQUENCE {
    eventCode INTEGER,
    eventIndex INTEGER,
    eventThreshold INTEGER,
    eventTime TimeStamp,
    eventDescr IA5STRING
}
```

The eventCode is a number which indicates the type of event. The eventCodes are defined below.

The eventIndex is an implementation specific value. It is considered good practice to make sure that a particular event is only generated in one place. It may be the case that certain HEMS generic events (for example, "no buffer space") may be generated by more than one place in an entity's code. To allow implementors and network managers to determine where the event is actually being generated, implementors should make sure that a distinct eventIndex is assigned to each location in the code that generates a particular event.

The eventThreshold is the value of the event threshold when the event was sent.

The eventTime indicates when the trap was generated.

The eventDescr is a text string which describes the event. This

description should explain the general problem (e.g., "no buffer space") and may also, optionally, include additional information about why this particular event was generated (e.g., "could not send ICMP redirect").

Event Definitions

The remainder of this memo presents a few generic events, which are presented for illustration only. Implementors interested in supporting events should contact the authors to help work out a more comprehensive set of definitions.

The format of the event definitions is:

EVENT CODE: The event code number.

Definition: Defines the event.

Related Objects: The list of related objects which *must* be returned following the event header. All objects should be returned as fully qualified objects (with ASN.1 codes tracing a complete path from the root object dictionary). If no objects are specified, then no related objects are required.

Event Status: Events are either required of all conforming implementations, required if the entity supports a particular feature (e.g., TCP events) or optional.

Notes: Any additional notes about the event.

List of Events

The next few event codes are for system (as opposed to more network oriented) events.

EVENT CODE: 0

Definition: Unused

EVENT CODE: 1

Definition: The entity has rebooted.

Related Objects: An INTEGER which is the highest HEMP

messageID reached by the trap system before the system crashed.

EVENT CODE: 2

Definition: The entity is about to go into test mode.

EVENT CODE: 3

Definition: The entity is about to reset.

EVENT CODE: 4

Definition: The entity is about to reboot.

EVENT CODE: 5

Definition: The entity is about to halt.

EVENT CODE: 6

Definition: The system is close to depleting its packet buffer space.

Event Status: optional

EVENT CODE: 7

Definition: The system has depleted its packet buffer space.

EVENT CODE: 8

Definition: The system has depleted a non-packet buffer space.

Note: The two trap codes above do not deal neatly with systems which have multiple buffer pools, each of which may be depleted separately, with very different effects on the entity.

The next set of event codes apply to events related to network interfaces.

EVENT CODE: 1024

Definition: The given interface has just come up.

Related Objects: The InterfaceData structure for the interface.

EVENT CODE: 1025

Definition: The given interface has just been taken down.

Related Objects: The InterfaceData structure for the interface.

EVENT CODE: 1026

Definition: The given interface has just gone into test mode.

Related Objects: The InterfaceData structure for the interface.

The next set of event codes are used to report IP-level errors.

EVENT CODE: 2048

Definition: Unable to route IP packet.

EVENT CODE: 2049

Definition: Bad IP checksum.

EVENT CODE: 2050

Definition: An IP packet with a bad header was received (for example, with a broadcast or multicast IP address as the source, or the wrong IP version number, or a header length which is too short).

Related Objects: Should return the IP header of the packet. Note that an IP header type has not yet been defined.

EVENT CODE: 2051

Definition: Packet for unsupported IP transport protocol.

Related Objects: Should return the IP header of the packet.
Note that an IP header type has not yet been defined.

EVENT CODE: 2052

Definition: A stunted IP packet was received (smaller than the IP length says it should be).

Related Objects: Should return the IP header of the packet.
Note that an IP header type has not yet been defined.

EVENT CODE: 2053

Definition: An oversize IP packet was received (larger than the IP length says it should be).

Related Objects: Should return the IP header of the packet.
Note that an IP header type has not yet been defined.

EVENT CODE: 2054

Definition: A good IP packet was discarded (usually to free up buffer space).

Related Objects: Should return the IP header of the packet.
Note that an IP header type has not yet been defined.

EVENT CODE: 2055

Definition: An IP packet's time-to-live as expired.

Related Objects: Should return the IP header of the packet.
Note that an IP header type has not yet been defined.

EVENT CODE: 2056

Definition: This IP fragment has timed out.

Related Objects: Should return the header of the fragment.
Note that an IP header type has not yet been defined.

AREAS FOR FURTHER STUDY

There are several parts of this document that could use additional study. Comments from readers are welcome.

The whole event system needs thorough examination. It is not clear that the event control mechanism strikes the proper balance between sufficient flexibility to allow monitoring centers to customize their event stream, and keeping the basic mechanism simple. Further, the problem of defining generic events for all entities is an immense task. Finally, the system of appending required values after traps, followed by optional values read from the data tree feels a bit cumbersome. It would be nice if all values were in the same data space.

Several readers have suggested it might make more sense to keep TCP connection parameters on a per-connection basis rather than globally.

The method for specifying the TCP round-trip time algorithm needs to be refined. The expression syntax should be sufficiently general that all round-trip-time-related algorithms (e.g., those for time or routing protocols) can be expressed in it.

Much more research could be done into what information needs to be gathered to effectively monitor a network.