<draft-ietf-entmib-entmib-01.txt>

Entity MIB

14 January 1996

Keith McCloghrie
Cisco Systems Inc.
kzm@cisco.com

Andy Bierman
Bierman Consulting
abierman@west.net

Status of this Memo

This document is an Internet-Draft. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet- Drafts as reference material or to cite them other than as ``work in progress.''

To learn the current status of any Internet-Draft, please check the ``1id-abstracts.txt'' listing contained in the Internet- Drafts Shadow Directories on ds.internic.net (US East Coast), nic.nordu.net (Europe), ftp.isi.edu (US West Coast), or munnari.oz.au (Pacific Rim).

1. Introduction

This memo defines an experimental portion of the Management Information Base (MIB) for use with network management protocols in the Internet community. In particular, it describes managed objects used for managing multiple logical entities managed by a single SNMP agent.

2. The SNMPv2 Network Management Framework

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

- RFC 1442 [1] which defines the SMI, the mechanisms used for describing and naming objects for the purpose of management.
- STD 17, RFC 1213 [2] defines MIB-II, the core set of managed objects for the Internet suite of protocols.
- RFC 1445 [3] which defines the administrative and other architectural aspects of the framework.
- RFC 1448 [4] which defines the protocol used for network access to managed objects.

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

2.1. Object Definitions

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

3. Overview

There is a need for a standardized way of representing a single agent which supports multiple instances of one MIB. This is already true for at least 3 standard MIBs, and is likely to become true for more and more MIBs as time passes. For example:

- multiple instances of a bridge supported within a single device having a single agent;

- multiple repeaters supported by a single agent;

- multiple OSPF backbone areas, each one operating as part of its own Autonomous System, and each identified by the same area-id (e.g., 0.0.0.0), supported inside a single router with one agent.

The fact that it is a single agent in each of these cases implies there is some relationship which binds all of these entities together. Effectively, there is some "overall" physical entity which houses the sum of the things managed by that one agent, i.e., there are multiple "logical" entities within a single physical entity. Sometimes, the overall physical entity contains multiple (smaller) physical entities and each logical entity is associated with a particular such physical entity. Sometimes, the overall physical entity is a "compound" of multiple physical entities (e.g., a stack of stackable hubs).

What is needed is a way to determine exactly what logical entities are managed by the agent (either by SNMPv1 or SNMPv2), and thereby to be able to communicate with the agent about a particular logical entity. When different logical entities are associated with different physical entities within the overall physical entity, it is also useful to be able to use this information to distinguish between logical entities.

In these situations, there is no need for varbinds for multiple logical entities to be referenced in the same SNMP message (although that might be useful in the future). Rather, it is sufficient, and in some situations preferable, to have the context/community in the message identify the logical entity to which the varbinds apply.

3.1. Terms

Some new terms are used throughout this document:

- Naming Scope
  A "naming scope" represents the set of information that may be potentially accessed through a single SNMP operation. All instances within the naming scope share the same unique identifier space. For SNMPv1, a naming scope is identified by the value of the associated 'entLogicalCommunity' instance.

- Logical Entity
  A managed system contains one or more logical entities, each represented by at most one instantiation of each of a particular set of MIB objects. A set of management functions is associated with each logical entity. Examples of logical entities include routers, bridges, print-servers, etc.

- Physical Entity
  A "physical entity" or "physical component" represents an identifiable physical resource within a managed system. Zero or more logical entities may utilize a physical resource at any given time. It is an implementation-specific manner as to which physical components are represented by an agent in the EntPhysicalTable. Typically, physical resources (e.g. communications ports, backplanes, sensors, daughter-cards, power supplies, the overall chassis) which can be managed via functions associated with one or more logical entities are included in the MIB.

- Containment Tree
  Each physical component may optionally be modeled as 'contained' within another physical component. A "containment-tree" is the conceptual sequence of entPhysicalIndex values which uniquely specifies the exact physical location of a physical component within the managed system. It is generated by 'following and recording' each 'entPhysicalContainedIn' instance until a value of zero (indicating no further containment) is found.

  It is suggested that physical containment-trees retain their identity across reboots. Specifically, two identical hardware configurations should produce the same set of containment-trees for every corresponding entry in the entPhysicalTable (i.e. the same set of entPhysicalEntries with the same entPhysicalIndex values,

  [ed. want to say anything about container/containee guidelines here?]

3.2. Relationship to Community Strings

For SNMPv1, distinguishing between different logical entities is one (but not the only) purpose of the community string [6]. This is accommodated by representing each community string as a logical entity.

Note that different logical entities may 'share' the same naming scope (and therefore the same values of entLogicalCommunity). In such a case, individual logical entities can be identified by examining the sysORTable within the same naming scope.

3.3. Relationship to Proxy Mechanisms

The Entity MIB is designed to allow functional component discovery. The administrative relationships between different logical entities are not visible in any Entity MIB tables.

The management of administrative framework functions is not an explicit goal of the Entity MIB WG at this time. This new area of functionality may be revisited after some operational experience with the Entity MIB is gained.

3.4. Relationship to a Chassis MIB

Some readers may recall that a previous IETF working group attempted to define a Chassis MIB. No consensus was reached by that working group, possibly because its scope was too broad. As such, it is not the purpose of this MIB to be a "Chassis MIB replacement", nor is it within the scope of this MIB to contain all the information which might be necessary to manage a "chassis". On the other hand, the entities represented by an implementation of this MIB might well be contained in a chassis.

3.5. Relationship to the Interfaces MIB

The Entity MIB contains a mapping table identifying physical components that have 'external values' (e.g. ifIndex) associated with them within a given naming scope. This table can be used to identify the physical location of each interface in the ifTable. Since ifIndex values in different contexts are not related to one another, the interface to physical component associations are relative to a specific logical entity within the agent.

[ed. say anything about entAliasMappingTable and ifIndex renumbering?]

3.6. Relationship to the Other MIBs

The Entity MIB contains a mapping table identifying physical components that have identifiers from other standard MIBs associated with them. For example, this table can be used along with the physical mapping table to identify the physical location of each repeater port in the rptrPortTable, each bridge port in the dot1dBasePortTable, or each ifIndex in the ifTable.

3.7. Re-Configuration of Entities

All the MIB objects defined in this MIB have at most a read-only MAX-ACCESS clause, i.e., none are write-able. This is another conscious decision by the authors to limit this MIB's scope. It is possible that this restriction could be lifted after implementation experience.

3.8. MIB Structure

This MIB contains five tables: the entPhysicalTable and the entLogicalTable each provide a list of entities. The entLPMappingTable provides mappings between logical and physical entities. The entAliasMappingTable provides mappings between physical components and associated identifiers from other MIBs. For example, a physical repeater port may be associated with an instance of 'rptrPortGroupIndex.1.5', or 'ifIndex.12', or both. The entPhysicalContainsTable provides efficient discovery of the containment relationships of all physical entities (also derivable from 'entPhysicalContainedIn' values).

The entPhysicalTable contains one row per physical entity, and should always contains at least one row for an "overall" physical entity. Each row is indexed by an arbitrary, small integer, and contains a description and type of the physical entity. It also optionally contains the index number of another row in the same table indicating a containment relationship between the two.

The entLogicalTable contains one row per logical entity. Each row is indexed by an arbitrary, small integer and contains a name, description, and type of the logical entity. It also contains information to allow

SNMPv2 and/or SNMPv1 access to the MIB information for the logical entity.

The entLPMappingTable contains mappings between entLogicalIndex values (logical entities) and entPhysicalIndex values (the physical components supporting that entity). A logical entity can map to more than one physical component, and more than one logical entity can map to (share) the same physical component.

The entAliasMappingTable contains mappings between entPhysicalIndex, entLogicalIndex pairs and 'alias' object identifier values. This allows resources managed with other MIBs (e.g. repeater ports, bridge ports, physical and logical interfaces) to be identified in the physical entity hierarchy. Note that each alias identifier is only relevant in a particular naming scope.

The entPhysicalContainsTable contains simple mappings between 'entPhysicalContainsIn' values for each container/containee relationship in the managed system. The indexing of this table allows an NMS to quickly discover the 'entPhysicalIndex' values for all children of a given physical entity.

3.9. Multiple Agents

Even though a primary motivation for this MIB is to represent the multiple logical entities supported by a single agent, it is also possible to use it to represent multiple logical entities supported by multiple agents (in the same "overall" physical entity). Indeed, it is implicit in the SNMP architecture, that the number of agents is transparent to a network management station.

4. Definitions

```
ENTITY-MIB DEFINITIONS ::= BEGIN

IMPORTS
    MODULE-IDENTITY, OBJECT-TYPE, experimental,
    IpAddress
        FROM SNMPv2-SMI
    DisplayString, AutonomousType, TruthValue
        FROM SNMPv2-TC
    Context
        FROM SNMPv2-PARTY-MIB
    MODULE-COMPLIANCE, OBJECT-GROUP
        FROM SNMPv2-CONF;


entityMIB MODULE-IDENTITY
    LAST-UPDATED "9601020000Z"
    ORGANIZATION "IETF ENTMIB Working Group"
    CONTACT-INFO
            "       Keith McCloghrie
                    Cisco Systems Inc.
                    170 West Tasman Drive
                    San Jose, CA 95134
                    408-526-5260
                    kzm@cisco.com

                    Andy Bierman
                    Bierman Consulting
                    1200 Sagamore Lane
                    Ventura, CA 93001
                    805-648-2028
                    abierman@west.net"
    DESCRIPTION
            "The MIB module for representing multiple logical
            entities supported by a single SNMP agent."
    ::= { experimental xx }

entityMIBObjects OBJECT IDENTIFIER ::= { entityMIB 1 }
```

```
--           The Physical Entity Table

entPhysicalTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF EntPhysicalEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
            "This table contains one row per physical entity.  There is
            always at least one row for an 'overall' physical entity."
    ::= { entityMIBObjects 1 }

entPhysicalEntry       OBJECT-TYPE
    SYNTAX      EntPhysicalEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
            "Information about a particular physical entity.  An agent
            is expected to represent physical components in as much
            detail as possible.  If more than one agent within a chassis
            implements the Entity MIB, the information retrieved from
            each agent must be consistent, but not necessarily
            identical."
    INDEX   { entPhysicalIndex }
    ::= { entPhysicalTable 1 }

EntPhysicalEntry ::= SEQUENCE {
      entPhysicalIndex          INTEGER,
      entPhysicalDescr          DisplayString,
      entPhysicalVendorType     AutonomousType,
      entPhysicalContainedIn    INTEGER,
      entPhysicalClass          INTEGER
}

entPhysicalIndex    OBJECT-TYPE
    SYNTAX      INTEGER (1..2147483647)
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
            "The value of this object uniquely identifies the physical
            entity.  The value is a small positive integer; index values
            for different physical entities are not necessarily
            contiguous."
    ::= { entPhysicalEntry 1 }

entPhysicalDescr OBJECT-TYPE
```

```
    SYNTAX      DisplayString
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
            "A textual description of physical entity."
    ::= { entPhysicalEntry 2 }

entPhysicalVendorType OBJECT-TYPE
    SYNTAX      AutonomousType
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
            "An indication of the vendor-specific hardware type of the
            physical entity. Note that this is different from the
            definition of MIB-II's sysObjectID.

            An agent should set this object to a enterprise-specific
            registration identifier value indicating the specific
            equipment type in detail.  The associated instance of the
            entPhysicalClass object should be used to indicate the
            general type of hardware device.

            If no vendor-specific registration identifier exists for
            this physical entity, then the value { 0 0 } is returned. If
            the value is unknown by this agent, then the special value
            'entPClassUnknown' is returned."
    ::= { entPhysicalEntry 3 }

entPhysicalContainedIn OBJECT-TYPE
    SYNTAX      INTEGER (0..2147483647)
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
            "The value of entPhysicalIndex for the physical entity which
            'contains' this physical entity.  A value of zero indicates
            this physical entity is not contained in any other physical
            entity.  Note that the set of 'containment' relationships
            define a strict hierarchy; that is, recursion is not
            allowed."
    ::= { entPhysicalEntry 4 }


entPhysicalClass OBJECT-TYPE
    SYNTAX      INTEGER  {
        other(1),
```

```
        unknown(2),
        chassis(3),
        backplane(4),
        container(5),   -- slot or daughter-card holder
        powerSupply(6),
        fan(7),
        sensor(8),
        module(9),
        port(10)
        -- some others here that I forgot?
    }
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
            "An indication of the general hardware type of the physical
            entity.

            An agent should set this object to the standard enumeration
            value which most accurately indicates the general class of
            the physical entity, or the primary class if there is more
            than one.

            If no appropriate standard registration identifier exists
            for this physical entity, then the value 'other(1)' is
            returned. If the value is unknown by this agent, then the
            value 'unknown(2)' is returned."
    ::= { entPhysicalEntry 5 }
```

```
--           The Logical Entity Table
entLogicalTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF EntLogicalEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
            "This table contains one row per logical entity."
    ::= { entityMIBObjects 2 }

entLogicalEntry       OBJECT-TYPE
    SYNTAX      EntLogicalEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
            "Information about a particular logical entity.  Entities
            may be managed by this agent or other SNMP agents (possibly)
            in the same chassis."
    INDEX       { entLogicalIndex }
    ::= { entLogicalTable 1 }

EntLogicalEntry ::= SEQUENCE {
      entLogicalIndex            INTEGER,
      entLogicalDescr            DisplayString,
      entLogicalType             AutonomousType,
      entLogicalCommunity        OCTET STRING,
      entLogicalIpAddress        IpAddress
}

entLogicalIndex OBJECT-TYPE
    SYNTAX      INTEGER (1..2147483647)
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
            "The value of this object uniquely identifies the logical
            entity. The value is a small positive integer; index values
            for different logical entities are are not necessarily
            contiguous."
    ::= { entLogicalEntry 1 }
```

```
entLogicalDescr OBJECT-TYPE
    SYNTAX      DisplayString
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
            "A textual description of logical entity."
    ::= { entLogicalEntry 2 }

entLogicalType OBJECT-TYPE
    SYNTAX      AutonomousType
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
            "An indication of the type of logical entity.  This will
            typically be the OBJECT IDENTIFIER name of the node in the
            SMI's naming hierarchy which represents the major MIB
            module, or the majority of the MIB modules, supported by the
            logical entity.  For example:
               a logical entity of a regular host/router -> mib-2
               a logical entity of a 802.1d bridge     -> dot1dBridge
               a logical entity of a 802.3 repeater  ->
            snmpDot3RptrMgmt"
    ::= { entLogicalEntry 3 }

entLogicalCommunity OBJECT-TYPE
    SYNTAX      OCTET STRING
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
            "A SNMPv1 community-string which can be used to access
            detailed management information for this logical entity.
            The agent should allow read access with this community
            string (to an appropriate subset of all managed objects) and
            may also choose to return a community string based on the
            privileges of the request used to read this object (e.g.
            only return a string having read-write privileges when
            accessed with read-write privileges).

            A conformant SNMP agent may wish to conserve naming scopes
            by representing multiple logical entities in a single 'main'
            naming scope.  This is possible when the logical entities
            represented by the same value of entLogicalCommunity have no
            object instances in common.  For example, 'bridge1' and
            'repeater1' may be part of the main naming scope, but two
            additional community strings are needed to represent
```

```
            'bridge2' and 'repeater2'.

            Logical entities 'bridge1' and 'repeater1' would be
            represented by sysOREntries associated with the 'main'
            naming scope.

            For agents not accessible via SNMPv1, the value of this
            object is the empty-string."
    ::= { entLogicalEntry 4 }

entLogicalIpAddress OBJECT-TYPE
    SYNTAX      IpAddress
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
            "The IP-address of the SNMPv1 agent from which detailed
            management information for this logical entity can be
            obtained.  For agents not accessible via SNMPv1, the value
            of this object is 0.0.0.0."
    ::= { entLogicalEntry 5 }
```

```
--   entLPMappingTable: for each entity index, there are N
--      rows, each representing a physical component
--      that is associated with the indicated entity
--
-- entity to component table
entLPMappingTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF EntLPMappingEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
            "This table contains zero or more rows of logical entity to
            physical equipment associations."
    ::= { entityMIBObjects 3 }

entLPMappingEntry       OBJECT-TYPE
    SYNTAX      EntLPMappingEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
            "Information about a particular logical entity to physical
            equipment binding."
    INDEX       { entLogicalIndex, entLPPhysicalIndex }
    ::= { entLPMappingTable 1 }

EntLPMappingEntry ::= SEQUENCE {
      entLPPhysicalIndex         INTEGER
}

entLPPhysicalIndex OBJECT-TYPE
    SYNTAX      INTEGER (1..2147483647)
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
            "The value of this object identifies a particular
            entPhysicalEntry associated with the indicated
            entLogicalEntity."
    ::= { entLPMappingEntry 1 }
```

```
-- logical entity/component to alias table
entAliasMappingTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF EntAliasMappingEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
            "This table contains zero or more rows of logical entity,
            and physical component to external identifier associations."
    ::= { entityMIBObjects 4 }

entAliasMappingEntry       OBJECT-TYPE
    SYNTAX      EntAliasMappingEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
            "Information about a particular physical equipment, logical
            entity to external identifier binding. Note that the same
            physical component-logical entity pair may have an arbitrary
            number of external identifier (alias) bindings."
    INDEX { entLogicalIndex, entPhysicalIndex, entAliasMappingIndex }
    ::= { entAliasMappingTable 1 }

EntAliasMappingEntry ::= SEQUENCE {
      entAliasMappingIndex      INTEGER,
      entAliasIdentifier        OBJECT IDENTIFIER
}

entAliasMappingIndex OBJECT-TYPE
    SYNTAX      INTEGER (1..2147483647)
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
            "The value of this object uniquely identifies the particular
            binding for a specific logical entity on a particular
            physical component. The value is a small positive integer;
            index values for different entAlias mappings are not
            necessarily contiguous."
    ::= { entAliasMappingEntry 1 }

entAliasIdentifier OBJECT-TYPE
    SYNTAX      OBJECT IDENTIFIER
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
            "The value of this object identifies a particular MIB
```

```
            instance associated with the indicated entPhysicalEntry and
            entLogicalEntry pair.

            For example, suppose a physical port was represented by
            entPhysicalEntry.3, and entLogicalEntry.1 existed for a
            repeater, entLogicalEntry.2 existed for a bridge, and the
            bridge port was also represented in the ifTable. Then there
            might be three associated instances of entAliasIdentifier:
               entAliasIdentifier.3.1.1 == rptrPortGroupIndex.5.2
               entAliasIdentifier.3.2.1 == dot1dBasePort.2
               entAliasIdentifier.3.2.2 == ifIndex.2 "
    ::= { entAliasMappingEntry 2 }
```

```
-- physical mapping table
entPhysicalContainsTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF EntPhysicalContainsEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
            "A table which exposes the container/containee relationships
            between physical entities."
    ::= { entityMIBObjects 5 }

entPhysicalContainsEntry OBJECT-TYPE
    SYNTAX      EntPhysicalContainsEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
            "A single container/containee relationship."
    INDEX       { entPhysicalIndex, entPhysicalChildIndex }
    ::= { entPhysicalContainsTable 1 }

EntPhysicalContainsEntry ::= SEQUENCE {
      entPhysicalChildIndex INTEGER
}

entPhysicalChildIndex OBJECT-TYPE
    SYNTAX      INTEGER (1..2147483647)
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
            "The value of entPhysicalIndex for the contained physical
            entity."
    ::= { entPhysicalContainsEntry 1 }
```

```
-- last change timestamp for the whole MIB
entLastConfigChangeTime OBJECT-TYPE
    SYNTAX      Timestamp
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
            "The value of sysUpTime at the time any of these events
            occur:
                * a conceptual row is created or deleted in any of these
tables:
                    - entPhysicalTable
                    - entLogicalTable
                    - entLPMappingTable
                    - entAliasMappingTable
                    - entPhysicalContainsTable

                * any instance in the following list of objects changes value:
                    - entPhysicalDescr
                    - entPhysicalVendorType
                    - entPhysicalContainedIn
                    - entPhysicalClass
                    - entLogicalDescr
                    - entLogicalType
                    - entLogicalCommunity
                    - entLogicalIpAddress
                    - entAliasIdentifier
            "
    ::= { entityMIBObjects 6 }
```

```
-- Entity MIB Trap Definitions
entityMIBTraps    OBJECT IDENTIFIER ::= { entityMIB 2 }

entConfigChange NOTIFICATION-TYPE
    OBJECTS {
            entLastChangeTime
            }
    STATUS             current
    DESCRIPTION
            "An entConfigChange trap is sent when the value of
            entLastConfigChangeTime changes. It can be utilized by an
            NMS to trigger logical/physical entity table maintenance
            polls. This trap is throttled by the agent.

            The value of entLastChangeTime at the time the config-
            change-event is generated by the agent is encoded at the
            only var-bind in the trap.

            An agent must take care not to generate more than one
            entLastChangeTime 'trap-event' in a five second period (a
            'trap-event' is the transmission of a single trap PDU to a
            list of trap receivers).  If additional configuration
            changes occur within the five second 'throttling' period,
            then the agent should discard all but the most recent trap-
            event (if any), rather than queueing them and generating
            trap-events (one every five seconds) in sequence. "
   ::= { entityMIBTraps 1 }
```

```
-- conformance information
entityConformance OBJECT IDENTIFIER ::= { entityMIB 3 }

entityCompliances OBJECT IDENTIFIER ::= { entityConformance 1 }
entityGroups      OBJECT IDENTIFIER ::= { entityConformance 2 }

-- compliance statements

entityCompliance MODULE-COMPLIANCE
    STATUS  current
    DESCRIPTION
            "The compliance statement for SNMPv2 entities
            which implement the Entity MIB."
    MODULE  -- this module
        MANDATORY-GROUPS { entityGroup }
    ::= { entityCompliances 1 }


-- units of conformance

entityGroup    OBJECT-GROUP
    OBJECTS { entPhysicalDescr,
              entPhysicalVendorType,
              entPhysicalContainedIn,
              entPhysicalClass,
              entLogicalDescr,
              entLogicalType,
              entLogicalCommunity,
              entLogicalIpAddress,
              entLPPhysicalIndex,
              entAliasIdentifier,
              entPhysicalChildIndex,
              entLastChangeTime,
              entConfigChange -- separate conformance group needed for trap??
            }
    STATUS  current
    DESCRIPTION
            "The collection of objects which are used to
            represent the multiple logical entities,
            physical components, interfaces, and port
            alias identifiers for which a single agent
            provides MIB information."
    ::= { entityGroups 1 }

END
```

5. Usage Examples

5.1. Router/Bridge

A bi-lingual (SNMPv1 and SNMPv2) router containing two slots. Each slot contains a 3 port router/bridge module. Each port is represented in the ifTable. There are two logical instances of OSPF running and two logical bridges:

```
  Physical entities -- entPhysicalTable:
    1 Field-replaceable physical chassis:
      entPhysicalDescr.1 ==             "Acme Chassis Model 100"
      entPhysicalVendorType.1  ==       acmeProducts.chassisTypes.1
      entPhysicalContainedIn.1 ==       0
      entPhysicalClass.1 ==             chassis(3)

    2 slots within the chassis:
      entPhysicalDescr.2 ==             "Acme Router Chassis Slot 1"
      entPhysicalVendorType.2  ==       acmeProducts.slotTypes.1
      entPhysicalContainedIn.2 ==       1
      entPhysicalClass.2 ==             container(5)

      entPhysicalDescr.3 ==             "Acme Router Chassis Slot 2"
      entPhysicalVendorType.3  ==       acmeProducts.slotTypes.1
      entPhysicalContainedIn.3 ==       1
      entPhysicalClass.3 ==             container(5)

    2 Field-replaceable modules:
    Slot 1 contains a module with 3 ports:
      entPhysicalDescr.4 ==             "Acme Router Module Model 10"
      entPhysicalVendorType.4  ==       acmeProducts.moduleTypes.14
      entPhysicalContainedIn.4 ==       2
      entPhysicalClass.4 ==             module(9)

      entPhysicalDescr.5 ==             "Acme Router Ethernet Port 1"
      entPhysicalVendorType.5  ==       acmeProducts.portTypes.2
      entPhysicalContainedIn.5 ==       4
      entPhysicalClass.5 ==             port(10)

      entPhysicalDescr.6 ==             "Acme Router Ethernet Port 2"
      entPhysicalVendorType.6  ==       acmeProducts.portTypes.2
      entPhysicalContainedIn.6 ==       4
      entPhysicalClass.6 ==             port(10)

      entPhysicalDescr.7 ==             "Acme Router Fddi Port 3"
```

```
      entPhysicalVendorType.7  ==       acmeProducts.portTypes.3
      entPhysicalContainedIn.7 ==       4
      entPhysicalClass.7 ==             port(10)

   Slot 2 contains another 3-port module:
      entPhysicalDescr.8 ==             "Acme Router Module Model 11"
      entPhysicalVendorType.8  ==       acmeProducts.moduleTypes.15
      entPhysicalContainedIn.8 ==       3
      entPhysicalClass.8 ==             module(9)

      entPhysicalDescr.9 ==             "Acme Router Fddi Port 1"
      entPhysicalVendorType.9 ==        acmeProducts.portTypes.3
      entPhysicalContainedIn.9 ==       8
      entPhysicalClass.9 ==             port(10)

      entPhysicalDescr.10 ==            "Acme Router Ethernet Port 2"
      entPhysicalVendorType.10 ==       acmeProducts.portTypes.2
      entPhysicalContainedIn.10 ==      8
      entPhysicalClass.10 ==            port(10)

      entPhysicalDescr.11 ==            "Acme Router Ethernet Port 3"
      entPhysicalVendorType.11 ==       acmeProducts.portTypes.2
      entPhysicalContainedIn.11 ==      8
      entPhysicalClass.11 ==            port(10)

   Logical entities -- entLogicalTable
   2 OSPF instances:
      entLogicalDescr.1 ==           "ospf-1"
      entLogicalType.1 ==            ospf
      entLogicalCommunity.1 ==       "public-ospf1"
      entLogicalIpAddress.1 ==       124.125.126.127

      entLogicalDescr.2 ==           "ospf-2"
      entLogicalType.2 ==            ospf
      entLogicalCommunity.2 ==       "public-ospf2"
      entLogicalIpAddress.2 ==       124.125.126.127

    2 logical bridges:
      entLogicalDescr.3 ==           "bridge1"
      entLogicalType.3  ==           dod1dBridge
      entLogicalCommunity.3 ==       "public-bridge1"
      entLogicalIpAddress.3 ==        124.125.126.127

      entLogicalDescr.4 ==           "bridge2"
      entLogicalType.4 ==            dod1dBridge
```

```
      entLogicalCommunity.4 ==        "public-bridge2"
      entLogicalIpAddress.4 ==        124.125.126.127

Logical to Physical Mappings:
  1st OSPF instance: uses module 1-port 1
      entLPPhysicalIndex.1.5 ==         5

  2nd OSPF instance: uses module 2-port 1
      entLPPhysicalIndex.2.9 ==         9

  1st bridge group: uses module 1, all ports
      entLPPhysicalIndex.3.5 ==         5
      entLPPhysicalIndex.3.6 ==         6
      entLPPhysicalIndex.3.7 ==         7

  2nd bridge group: uses module 2, all ports
      entLPPhysicalIndex.4.9  ==        9
      entLPPhysicalIndex.4.10 ==        10
      entLPPhysicalIndex.4.11 ==        11

Logical to Physical to Alias Mappings -- entAliasMappingTable:
  Bridge 1 uses Ports 1..3 on Slot 1
      entAliasIdentifier.3.5.1 ==       dot1dBasePort.1
      entAliasIdentifier.3.5.2 ==       ifIndex.1
      entAliasIdentifier.3.6.1 ==       dot1dBasePort.2
      entAliasIdentifier.3.6.2 ==       ifIndex.2
      entAliasIdentifier.3.7.1 ==       dot1dBasePort.3
      entAliasIdentifier.3.7.2 ==       ifIndex.3

  Bridge 2 uses Ports 1..3 on Slot 2; uses same Bridge MIB and ifIndex
  values as Bridge 1, but in a different context, and representing
  different physical ports:
      entAliasIdentifier.4.9.1 ==       dot1dBasePort.1
      entAliasIdentifier.4.9.2 ==       ifIndex.1
      entAliasIdentifier.4.10.1 ==      dot1dBasePort.2
      entAliasIdentifier.4.10.2 ==      ifIndex.2
      entAliasIdentifier.4.11.1 ==      dot1dBasePort.3
      entAliasIdentifier.4.11.2 ==      ifIndex.3

Physical Containment Tree -- entPhysicalContainsTable
  chassis has two containers:
      entPhysicalChildIndex.1.2 = 2
      entPhysicalChildIndex.1.3 = 3

  container 1 has a module:
```

```
      entPhysicalChildIndex.2.4 = 4

  container 2 has a module
      entPhysicalChildIndex.3.8 = 8

  module 1 has some ports:
      entPhysicalChildIndex.4.5 = 5
      entPhysicalChildIndex.4.6 = 6
      entPhysicalChildIndex.4.7 = 7

  module 2 has some ports:
      entPhysicalChildIndex.8.9 = 9
      entPhysicalChildIndex.8.10 = 10
      entPhysicalChildIndex.1.11 = 11
```

5.2. Repeaters

An SNMPv1 only, 3-slot Hub with 2 backplane ethernet segments. Slot three is empty, and the remaining slots contain ethernet repeater modules.

```
   Physical entities -- entPhysicalTable:
    1 Field-replaceable physical chassis:
      entPhysicalDescr.1 ==          "Acme Repeater Chassis Model 110"
      entPhysicalVendorType.1 ==     acmeProducts.chassisTypes.2
      entPhysicalContainedIn.1 ==    0
      entPhysicalClass.1 ==          chassis(3)

    2 Chassis Ethernet Backplanes:
      entPhysicalDescr.2 ==          "Ethernet Backplane 1"
      entPhysicalVendorType.2 ==     acmeProducts.backplaneTypes.1
      entPhysicalContainedIn.2 ==    1
      entPhysicalClass.2 ==          backplane(4)

      entPhysicalDescr.3 ==          "Ethernet Backplane 2"
      entPhysicalVendorType.3  ==    acmeProducts.backplaneTypes.1
      entPhysicalContainedIn.3 ==    1
      entPhysicalClass.3 ==          backplane(4)

     3 slots within the chassis:
      entPhysicalDescr.4 ==          "Acme Hub Slot 1"
      entPhysicalVendorType.4  ==    acmeProducts.slotTypes.5
      entPhysicalContainedIn.4 ==    1
      entPhysicalClass.4 ==          container(5)
```

```
      entPhysicalDescr.5 ==          "Acme Hub Slot 2"
      entPhysicalVendorType.5  ==    acmeProducts.slotTypes.5
      entPhysicalContainedIn.5 ==    1
      entPhysicalClass.5 ==          container(5)

      entPhysicalDescr.6 ==          "Acme Hub Slot 3"
      entPhysicalVendorType.6  ==    acmeProducts.slotTypes.5
      entPhysicalContainedIn.6 ==    1
      entPhysicalClass.6 ==          container(5)

    Slot 1 contains a plug-in module with 4 10-BaseT ports:
      entPhysicalDescr.7  ==         "10Base-T Module Model 14"
      entPhysicalVendorType.7   ==   acmeProducts.moduleTypes.32
      entPhysicalContainedIn.7  ==   4
      entPhysicalClass.7 ==          module(9)

      entPhysicalDescr.8  ==         "10Base-T Port 1"
      entPhysicalVendorType.8   ==   acmeProducts.portTypes.10
      entPhysicalContainedIn.8  ==   7
      entPhysicalClass.8 ==          port(10)

      entPhysicalDescr.9  ==         "10Base-T Port 2"
      entPhysicalVendorType.9   ==   acmeProducts.portTypes.10
      entPhysicalContainedIn.9  ==   7
      entPhysicalClass.9 ==          port(10)

      entPhysicalDescr.10 ==         "10Base-T Port 3"
      entPhysicalVendorType.10  ==   acmeProducts.portTypes.10
      entPhysicalContainedIn.10 ==   7
      entPhysicalClass.10 ==         port(10)

      entPhysicalDescr.11 ==         "10Base-T Port 4"
      entPhysicalVendorType.11  ==   acmeProducts.portTypes.10
      entPhysicalContainedIn.11 ==   7
      entPhysicalClass.11 ==         port(10)

   Slot 2 contains another ethernet module with 2 ports.
      entPhysicalDescr.12 ==         "Acme 10Base-T Module Model 4"
      entPhysicalVendorType.12 ==    acmeProducts.moduleTypes.30
      entPhysicalContainedIn.12 =    5
      entPhysicalClass.12 ==         module(9)

      entPhysicalDescr.13 ==         "802.3 AUI Port 1"
      entPhysicalVendorType.13  ==   acmeProducts.portTypes.11
      entPhysicalContainedIn.13 ==   12
```

```
      entPhysicalClass.13 ==         port(10)

      entPhysicalDescr.14 ==          "10Base-T Port 2"
      entPhysicalVendorType.14  ==    acmeProducts.portTypes.10
      entPhysicalContainedIn.14 ==    12
      entPhysicalClass.14 ==          port(10)

   Logical entities -- entLogicalTable
     Repeater 1--comprised of any ports attached to backplane 1
       entLogicalDescr.1 ==         "repeater1"
       entLogicalType.1  ==         snmpDot3RptrMgt
       entLogicalCommunity.1        "public-repeater1"
       entLogicalIpAddress.1        124.125.126.128

     Repeater 2--comprised of any ports attached to backplane 2:
       entLogicalDescr.2 ==         "repeater2"
       entLogicalType.2  ==         snmpDot3RptrMgt
       entLogicalCommunity.2 ==     "public-repeater2"
       entLogicalIpAddress.2 ==     124.125.126.128

Logical to Physical Mappings -- entLPMappingTable:

  repeater1 uses backplane 1, slot 1-ports 1 & 2, slot 2-port 1
      entLPPhysicalIndex.1.2 ==         2
      entLPPhysicalIndex.1.8 ==         8
      entLPPhysicalIndex.1.9 ==         9
      entLPPhysicalIndex.1.13 ==        13

  repeater2 uses backplane 2, slot 1-ports 3 & 4, slot 2-port 2
      entLPPhysicalIndex.2.3 ==         3
      entLPPhysicalIndex.2.10 ==        10
      entLPPhysicalIndex.2.11 ==        11
      entLPPhysicalIndex.2.14 ==        14

Logical to Physical to Alias Mappings -- entAliasMappingTable:
  repeater1 uses slot 1-ports 1 & 2, slot 2-port 1
      entAliasIdentifier.1.8.1 ==       rptrPortGroupIndex.1.1
      entAliasIdentifier.1.9.1 ==       rptrPortGroupIndex.1.2
      entAliasIdentifier.1.13.1 ==      rptrPortGroupIndex.2.1

  repeater2 uses slot 1-ports 3 & 4, slot 2-port 2
      entAliasIdentifier.2.10.1 ==      rptrPortGroupIndex.1.3
      entAliasIdentifier.2.11.1 ==      rptrPortGroupIndex.1.4
      entAliasIdentifier.2.14.1 ==      rptrPortGroupIndex.2.2
```

Physical Containment Tree -- entPhysicalContainsTable

```
  chassis has two backplanes and three containers:
      entPhysicalChildIndex.1.2 = 2
      entPhysicalChildIndex.1.3 = 3
      entPhysicalChildIndex.1.4 = 4
      entPhysicalChildIndex.1.5 = 5
      entPhysicalChildIndex.1.6 = 6

  container 1 has a module:
      entPhysicalChildIndex.4.7 = 7

  container 2 has a module
      entPhysicalChildIndex.5.12 = 12
  -- container 3 is empty

  module 1 has some ports:
      entPhysicalChildIndex.7.8 = 8
      entPhysicalChildIndex.7.9 = 9
      entPhysicalChildIndex.7.10 = 10
      entPhysicalChildIndex.7.11 = 11

  module 2 has some ports:
      entPhysicalChildIndex.12.13 = 13
      entPhysicalChildIndex.12.14 = 14
```

6. Acknowledgements

This document was produced by the IETF Entity MIB Working Group.

7. References

[1] Case, J., McCloghrie, K., Rose, M., and S. Waldbusser, "Structure of Management Information for version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1442, SNMP Research,Inc., Hughes LAN Systems, Dover Beach Consulting, Inc., Carnegie Mellon University, April 1993.

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

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

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

[5] McCloghrie, K., and J. Galvin, "Party MIB for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1447, Hughes LAN Systems, Trusted Information Systems, April 1993.

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

[7] McCloghrie, K., and Kastenholtz, F., "Interfaces Group Evolution", RFC 1573, Hughes LAN Systems, FTP Software, January 1994.

8. Security Considerations

Security issues are not discussed in this memo.

9. Authors' Addresses

Keith McCloghrie
cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134
Phone: 408-526-5260
Email: kzm@cisco.com

Andy Bierman
Bierman Consulting
1200 Sagamore Lane
Ventura, CA 93001
Phone: 805-648-2028
Email: abierman@west.net

Table of Contents