Network Working Group
Request for Comments: 2037
Category: Standards Track

K. McCloghrie
A. Bierman
Cisco Systems

October 1996

Entity MIB using SMIv2

Status of this Memo

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

Table of Contents

1. Introduction

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

2. The SNMP Network Management Framework

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

- o the SMI, described in RFC 1902 [1], - the mechanisms used for describing and naming objects for the purpose of management.

- o the MIB-II, STD 17, RFC 1213 [2], - the core set of managed objects for the Internet suite of protocols.

- o the protocol, RFC 1157 [6] and/or RFC 1905 [4], - the protocol for accessing managed information.

Textual conventions are defined in RFC 1903 [3], and conformance statements are defined in RFC 1904 [5].

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

This memo specifies a MIB module that is compliant to the SNMPv2 SMI. A semantically identical MIB conforming to the SNMPv1 SMI can be produced through the appropriate translation.

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 presently 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 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.

- Multi-Scoped Object
  A MIB object, for which identical instance values identify different managed information in different naming scopes, is called a "multi-scoped" MIB object.

- Single-Scoped Object
  A MIB object, for which identical instance values identify the same managed information in different naming scopes, is called a "single-scoped" MIB object.

- 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 'up the tree towards the root', until a value of zero indicating no further containment is found.

Note that chassis slots, which are capable of accepting one or more module types from one or more vendors, are modeled as containers in this MIB. The value of entPhysicalContainedIn for a particular 'module' entity (entPhysicalClass value of 'module(9)') must be equal to an entPhysicalIndex that represents the parent 'container' entity (associated entPhysicalClass value of ('container(5)'). An agent must represent both empty and full containers in the entPhysicalTable.

3.2. Relationship to Community Strings

For community-based SNMP, 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). This is possible, providing they have no need for the same instance of a MIB object to represent different managed information.

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. An NMS cannot determine whether MIB instances in different naming scopes are realized locally or remotely (e.g. via some proxy mechanism) by examining any particular Entity MIB objects.

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.

Note that a network administrator will likely be able to associate community strings with naming scopes with proprietary mechanisms, as a matter of configuration. There are no mechanisms for managing naming scopes defined in this MIB.

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 [7]. Since ifIndex values in different contexts are not related to one another, the interface to physical component associations are relative to the same logical entity within the agent.

The Entity MIB also contains an 'entPhysicalName' object, which approximates the semantics of the ifName object from the Interfaces MIB [7] for all types of physical components.

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, or each interface in the ifTable.

3.7. Relationship to Naming Scopes

There is some question as to which MIB objects may be returned within a given naming scope. MIB objects which are not multi-scoped within a managed system are likely to ignore context information in implementation. In such a case, it is likely such objects will be returned in all naming scopes (e.g. not just the 'main' naming scope).

For example, a community string used to access the management information for logical device 'bridge2' may allow access to all the non-bridge related objects in the 'main' naming scope, as well as a second instance of the Bridge MIB.

It is an implementation-specific matter as to the isolation of single-scoped MIB objects by the agent. An agent may wish to limit the objects returned in a particular naming scope to just the multi-scoped objects in that naming scope (e.g. system group and the Bridge MIB). In this case, all single-scoped management information would belong to a common naming scope (e.g. 'main'), which itself may contain some multi-scoped objects (e.g. system group).

3.8. Multiple Instances of the Entity MIB

It is possible that more than one agent exists in a managed system, and in such cases, multiple instances of the Entity MIB (representing the same managed objects) may be available to an NMS.

In order to reduce complexity for agent implementation, multiple instances of the Entity MIB are not required to be equivalent or even consistent. An NMS may be able to 'align' instances returned by different agents by examining the columns of each table, but vendor-specific identifiers and (especially) index values are likely to be different. Each agent may be managing different subsets of the entire chassis as well.

When all of a physically-modular device is represented by a single agent, the entry for which entPhysicalContainedIn has the value zero would likely have 'chassis' as the value of its entPhysicalClass; alternatively, for an agent on a module where the agent represents only the physical entities on that module (not those on other modules), the entry for which entPhysicalContainedIn has the value zero would likely have 'module' as the value of its entPhysicalClass.

An agent implementation of the entLogicalTable is not required to contain information about logical entities managed primarily by other agents. That is, the entLogicalTAddress and entLogicalTDomain objects in the entLogicalTable are provided to support an historical multiplexing mechanism, not to identify other SNMP agents.

Note that the Entity MIB is a single-scoped MIB, in the event an agent represents the MIB in different naming scopes.

3.9. 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 a conscious decision by the working group to limit this MIB's scope. It is possible that this restriction could be lifted after implementation experience, by means of additional tables (using the AUGMENTS clause) for configuration and extended entity information.

3.10. MIB Structure

The Entity MIB contains five conformance groups:

- entityPhysical group
  Describes the physical entities managed by a single agent.

- entityLogical group
  Describes the logical entities managed by a single agent.

- entityMapping group
  Describes the associations between the physical entities, logical entities, interfaces, and non-interface ports managed by a single agent.

- entityGeneral group
  Describes general system attributes shared by potentially all types of entities managed by a single agent.

- entityNotifications group
  Contains status indication notifications.

3.10.1. entityPhysical Group

This group contains a single table to identify physical system components, called the entPhysicalTable.

The entPhysicalTable contains one row per physical entity, and must 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 entPhysicalEntry indicating a containment relationship between the two.

3.10.2. entityLogical Group

This group contains a single table to identify logical entities, called the entLogicalTable.

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 SNMPv1 or SNMPv2C [9] access to the MIB information for the logical entity.

3.10.3. entityMapping Group

This group contains a three tables to identify associations between different system components.

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 entLogicalIndex, entPhysicalIndex 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 'entPhysicalContainedIn' 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.

3.10.4. entityGeneral Group

This group contains general information relating to the other object groups.

At this time, the entGeneral group contains a single scalar object (entLastChangeTime), which represents the value of sysUptime when any part of the system configuration last changed.

3.10.5. entityNotifications Group

This group contains notification definitions relating to the overall status of the Entity MIB instantiation.

3.11. 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.

However, there is no agreement at this time as to the degree of cooperation which should be expected for agent implementations. Therefore, multiple agents within the same managed system are free to implement the Entity MIB independently. (Refer the section on "Multiple Instances of the Entity MIB" for more details).

4. Definitions

```
ENTITY-MIB DEFINITIONS ::= BEGIN

IMPORTS
    MODULE-IDENTITY, OBJECT-TYPE,
    mib-2, NOTIFICATION-TYPE
        FROM SNMPv2-SMI
    TDomain, TAddress, DisplayString, TEXTUAL-CONVENTION,
    AutonomousType, RowPointer, TimeStamp
        FROM SNMPv2-TC
    MODULE-COMPLIANCE, OBJECT-GROUP
        FROM SNMPv2-CONF;

entityMIB MODULE-IDENTITY
    LAST-UPDATED "9605160000Z"
    ORGANIZATION "IETF ENTMIB Working Group"
    CONTACT-INFO
            "        WG E-mail: entmib@cisco.com
                     Subscribe: majordomo@cisco.com
                                msg body: subscribe entmib

                     Keith McCloghrie
                     ENTMIB Working Group Chair
                     Cisco Systems Inc.
                     170 West Tasman Drive
                     San Jose, CA 95134
                     408-526-5260
                     kzm@cisco.com

                     Andy Bierman
                     ENTMIB Working Group Editor
                     Cisco Systems Inc.
                     170 West Tasman Drive
                     San Jose, CA 95134
                     408-527-3711
                     abierman@cisco.com"
    DESCRIPTION
            "The MIB module for representing multiple logical
            entities supported by a single SNMP agent."
    ::= { mib-2 47 }

entityMIBObjects OBJECT IDENTIFIER ::= { entityMIB 1 }

-- MIB contains four groups

entityPhysical OBJECT IDENTIFIER ::= { entityMIBObjects 1 }
entityLogical  OBJECT IDENTIFIER ::= { entityMIBObjects 2 }
```

```
entityMapping  OBJECT IDENTIFIER ::= { entityMIBObjects 3 }
entityGeneral  OBJECT IDENTIFIER ::= { entityMIBObjects 4 }


-- Textual Conventions
PhysicalIndex ::= TEXTUAL-CONVENTION
    STATUS          current
    DESCRIPTION
            "An arbitrary value which uniquely identifies the physical
            entity.  The value is a small positive integer; index values
            for different physical entities are not necessarily
            contiguous."
    SYNTAX          INTEGER (1..2147483647)


PhysicalClass ::= TEXTUAL-CONVENTION
    STATUS          current
    DESCRIPTION
            "An enumerated value which provides an indication of the
            general hardware type of a particular physical entity."
    SYNTAX      INTEGER  {
        other(1),
        unknown(2),
        chassis(3),
        backplane(4),
        container(5),   -- e.g. slot or daughter-card holder
        powerSupply(6),
        fan(7),
        sensor(8),
        module(9),      -- e.g. plug-in card or daughter-card
        port(10)
    }

--          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."
    ::= { entityPhysical 1 }

entPhysicalEntry       OBJECT-TYPE
    SYNTAX      EntPhysicalEntry
    MAX-ACCESS  not-accessible
    STATUS      current
```

```
    DESCRIPTION
            "Information about a particular physical entity.

            Each entry provides objects (entPhysicalDescr,
            entPhysicalVendorType, and entPhysicalClass) to help an NMS
            identify and characterize the entry, and objects
            (entPhysicalContainedIn and entPhysicalParentRelPos) to help
            an NMS relate the particular entry to other entries in this
            table."
    INDEX   { entPhysicalIndex }
    ::= { entPhysicalTable 1 }

EntPhysicalEntry ::= SEQUENCE {
      entPhysicalIndex          PhysicalIndex,
      entPhysicalDescr          DisplayString,
      entPhysicalVendorType     AutonomousType,
      entPhysicalContainedIn    INTEGER,
      entPhysicalClass          PhysicalClass,
      entPhysicalParentRelPos   INTEGER,
      entPhysicalName           DisplayString
}

entPhysicalIndex    OBJECT-TYPE
    SYNTAX      PhysicalIndex
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
            "The index for this entry."
    ::= { entPhysicalEntry 1 }

entPhysicalDescr OBJECT-TYPE
    SYNTAX      DisplayString
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
            "A textual description of physical entity.  This object
            should contain a string which identifies the manufacturer's
            name for the physical entity, and should be set to a
            distinct value for each version or model of the 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
            entPhysicalClass is used to indicate the general type of
            hardware device.

            If no vendor-specific registration identifier exists for
            this physical entity, or the value is unknown by this agent,
            then the value { 0 0 } 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      PhysicalClass
    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 }

entPhysicalParentRelPos OBJECT-TYPE
    SYNTAX      INTEGER (-1..2147483647)
```

```
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
            "An indication of the relative position of this 'child'
            component among all its 'sibling' components. Sibling
            components are defined as entPhysicalEntries which share the
            same instance values of each of the entPhysicalContainedIn
            and entPhysicalClass objects.

            An NMS can use this object to identify the relative ordering
            for all sibling components of a particular parent
            (identified by the entPhysicalContainedIn instance in each
            sibling entry).

            This value should match any external labeling of the
            physical component if possible. For example, for a module
            labeled as 'card #3', entPhysicalParentRelPos should have
            the value '3'.

            If the physical position of this component does not match
            any external numbering or clearly visible ordering, then
            user documentation or other external reference material
            should be used to determine the parent-relative position. If
            this is not possible, then the the agent should assign a
            consistent (but possibly arbitrary) ordering to a given set
            of 'sibling' components, perhaps based on internal
            representation of the components.

            If the agent cannot determine the parent-relative position
            for some reason, or if the associated value of
            entPhysicalContainedIn is '0', then the value '-1' is
            returned. Otherwise a non-negative integer is returned,
            indicating the parent-relative position of this physical
            entity.

            Parent-relative ordering normally starts from '1' and
            continues to 'N', where 'N' represents the highest
            positioned child entity.  However, if the physical entities
            (e.g. slots) are labeled from a starting position of zero,
            then the first sibling should be associated with a
            entPhysicalParentRelPos value of '0'.  Note that this
            ordering may be sparse or dense, depending on agent
            implementation.

            The actual values returned are not globally meaningful, as
            each 'parent' component may use different numbering
            algorithms. The ordering is only meaningful among siblings
            of the same parent component.
```

```
            The agent should retain parent-relative position values
            across reboots, either through algorithmic assignment or use
            of non-volatile storage."
    ::= { entPhysicalEntry 6 }


entPhysicalName OBJECT-TYPE
    SYNTAX      DisplayString
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
            "The textual name of the physical entity.  The value of this
            object should be the name of the component as assigned by
            the local device and should be suitable for use in commands
            entered at the device's `console'.  This might be a text
            name, such as `console' or a simple component number (e.g.
            port or module number), such as `1', depending on the
            physical component naming syntax of the device.

            If there is no local name, or this object is otherwise not
            applicable, then this object contains a zero-length string.

            Note that the value of entPhysicalName for two physical
            entities will be the same in the event that the console
            interface does not distinguish between them, e.g., slot-1
            and the card in slot-1."
    ::= { entPhysicalEntry 7 }

--           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.  At least
            one entry must exist."
    ::= { entityLogical 1 }

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,
      entLogicalTAddress         TAddress,
      entLogicalTDomain          TDomain
}

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 the logical entity.  This object
            should contain a string which identifies the manufacturer's
            name for the logical entity, and should be set to a distinct
            value for each version of the 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
            If an appropriate node in the SMI's naming hierarchy cannot
            be identified, the value 'mib-2' should be used."
    ::= { entLogicalEntry 3 }
```

```
entLogicalCommunity OBJECT-TYPE
    SYNTAX      OCTET STRING (SIZE (1..255))
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
            "An SNMPv1 or SNMPv2C 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.  Note that an agent may choose to return a community
            string with read-only privileges, even if this object is
            accessed with a read-write community string. However, the
            agent must take care not to return a community string which
            allows more privileges than the community string used to
            access this object.

            A compliant 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 at
            least one additional community string is 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 or SNMPv2C, the value
            of this object is the empty-string."
    ::= { entLogicalEntry 4 }

entLogicalTAddress OBJECT-TYPE
    SYNTAX      TAddress
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
            "The transport service address by which the logical entity
            receives network management traffic, formatted according to
            the corresponding value of entLogicalTDomain.

            For snmpUDPDomain, a TAddress is 6 octets long, the initial
            4 octets containing the IP-address in network-byte order and
            the last 2 containing the UDP port in network-byte order.
            Consult 'Transport Mappings for Version 2 of the Simple
```

```
            Network Management Protocol' (RFC 1906 [8]) for further
            information on snmpUDPDomain."
    ::= { entLogicalEntry 5 }

entLogicalTDomain OBJECT-TYPE
    SYNTAX      TDomain
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
            "Indicates the kind of transport service by which the
            logical entity receives network management traffic.
            Possible values for this object are presently found in the
            Transport Mappings for SNMPv2 document (RFC 1906 [8])."
    ::= { entLogicalEntry 6 }

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. For each logical entity
            known by this agent, there are zero or more mappings to the
            physical resources which are used to realize that logical
            entity.

            An agent should limit the number and nature of entries in
            this table such that only meaningful and non-redundant
            information is returned. For example, in a system which
            contains a single power supply, mappings between logical
            entities and the power supply are not useful and should not
            be included.

            Also, only the most appropriate physical component which is
            closest to the root of a particular containment tree should
            be identified in an entLPMapping entry.

            For example, suppose a bridge is realized on a particular
            module, and all ports on that module are ports on this
            bridge. A mapping between the bridge and the module would be
            useful, but additional mappings between the bridge and each
            of the ports on that module would be redundant (since the
            entPhysicalContainedIn hierarchy can provide the same
            information). If, on the other hand, more than one bridge
            was utilizing ports on this module, then mappings between
            each bridge and the ports it used would be appropriate.

            Also, in the case of a single backplane repeater, a mapping
```

```
            for the backplane to the single repeater entity is not
            necessary."
    ::= { entityMapping 1 }

entLPMappingEntry       OBJECT-TYPE
    SYNTAX      EntLPMappingEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
            "Information about a particular logical entity to physical
            equipment association. Note that the nature of the
            association is not specifically identified in this entry. It
            is expected that sufficient information exists in the MIBs
            used to manage a particular logical entity to infer how
            physical component information is utilized."
    INDEX       { entLogicalIndex, entLPPhysicalIndex }
    ::= { entLPMappingTable 1 }

EntLPMappingEntry ::= SEQUENCE {
      entLPPhysicalIndex         PhysicalIndex
}

entLPPhysicalIndex OBJECT-TYPE
    SYNTAX      PhysicalIndex
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
            "The value of this object identifies the index value of 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, representing
            mappings of logical entity and physical component to
            external MIB identifiers.  Each physical port in the system
            may be associated with a mapping to an external identifier,
            which itself is associated with a particular logical
            entity's naming scope. A 'wildcard' mechanism is provided to
            indicate that an identifier is associated with more than one
            logical entity."
    ::= { entityMapping 2 }
```

```
entAliasMappingEntry       OBJECT-TYPE
    SYNTAX      EntAliasMappingEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
            "Information about a particular physical equipment, logical
            entity to external identifier binding. Each logical
            entity/physical component pair may be associated with one
            alias mapping.  The logical entity index may also be used as
            a 'wildcard' (refer to the entAliasLogicalIndexOrZero object
            DESCRIPTION clause for details.)

            Note that only entPhysicalIndex values which represent
            physical ports (i.e. associated entPhysicalClass value is
            'port(10)') are permitted to exist in this table."
    INDEX { entPhysicalIndex, entAliasLogicalIndexOrZero }
    ::= { entAliasMappingTable 1 }

EntAliasMappingEntry ::= SEQUENCE {
      entAliasLogicalIndexOrZero        INTEGER,
      entAliasMappingIdentifier         RowPointer
}

entAliasLogicalIndexOrZero OBJECT-TYPE
    SYNTAX      INTEGER (0..2147483647)
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
            "The value of this object uniquely identifies the logical
            entity which defines the naming scope for the associated
            instance of the 'entAliasMappingIdentifier' object.

            If this object has a non-zero value, then it identifies the
            logical entity named by the same value of entLogicalIndex.

            If this object has a value of zero, then the mapping between
            the physical component and the alias identifier for this
            entAliasMapping entry is associated with all unspecified
            logical entities. That is, a value of zero (the default
            mapping) identifies any logical entity which does not have
            an explicit entry in this table for a particular
            entPhysicalIndex/entAliasMappingIdentifier pair.

            For example, to indicate that a particular interface (e.g.
            physical component 33) is identified by the same value of
            ifIndex for all logical entities, the following instance
            might exist:
```

```
                entAliasMappingIdentifier.33.0 = ifIndex.5

        In the event an entPhysicalEntry is associated differently
        for some logical entities, additional entAliasMapping
        entries may exist, e.g.:

                entAliasMappingIdentifier.33.0 = ifIndex.6
                entAliasMappingIdentifier.33.4 =  ifIndex.1
                entAliasMappingIdentifier.33.5 =  ifIndex.1
                entAliasMappingIdentifier.33.10 = ifIndex.12

        Note that entries with non-zero entAliasLogicalIndexOrZero
        index values have precedence over any zero-indexed entry. In
        this example, all logical entities except 4, 5, and 10,
        associate physical entity 33 with ifIndex.6."
    ::= { entAliasMappingEntry 1 }


entAliasMappingIdentifier OBJECT-TYPE
    SYNTAX      RowPointer
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The value of this object identifies a particular conceptual
        row associated with the indicated entPhysicalIndex and
        entLogicalIndex pair.

        Since only physical ports are modeled in this table, only
        entries which represent interfaces or ports are allowed.  If
        an ifEntry exists on behalf of a particular physical port,
        then this object should identify the associated 'ifEntry'.
        For repeater ports, the appropriate row in the
        'rptrPortGroupTable' should be identified instead.

        For example, suppose a physical port was represented by
        entPhysicalEntry.3, entLogicalEntry.15 existed for a
        repeater, and entLogicalEntry.22 existed for a bridge.  Then
        there might be two related instances of
        entAliasMappingIdentifier:
           entAliasMappingIdentifier.3.15 == rptrPortGroupIndex.5.2
           entAliasMappingIdentifier.3.22 == ifIndex.17
        It is possible that other mappings (besides interfaces and
        repeater ports) may be defined in the future, as required.

        Bridge ports are identified by examining the Bridge MIB and
        appropriate ifEntries associated with each 'dot1dBasePort',
        and are thus not represented in this table."
    ::= { 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. This table provides equivalent
            information found by constructing the virtual containment
            tree for a given entPhysicalTable but in a more direct
            format."
    ::= { entityMapping 3 }

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     PhysicalIndex
}

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

-- last change time stamp for the whole MIB
entLastChangeTime 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
                    - entPhysicalParentRelPos
                    - entPhysicalName
                    - entLogicalDescr
                    - entLogicalType
                    - entLogicalCommunity
                    - entLogicalTAddress
                    - entLogicalTDomain
                    - entAliasMappingIdentifier "
    ::= { entityGeneral 1 }

-- Entity MIB Trap Definitions
entityMIBTraps      OBJECT IDENTIFIER ::= { entityMIB 2 }
entityMIBTrapPrefix OBJECT IDENTIFIER ::= { entityMIBTraps 0 }

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

            An agent must not generate more than one entConfigChange
            'trap-event' in a five second period, where a 'trap-event'
            is the transmission of a single trap PDU to a list of trap
            destinations.  If additional configuration changes occur
            within the five second 'throttling' period, then these
            trap-events should be suppressed by the agent. An NMS should
            periodically check the value of entLastChangeTime to detect
            any missed entConfigChange trap-events, e.g. due to
            throttling or transmission loss."
   ::= { entityMIBTrapPrefix 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 SNMP entities which implement
            the Entity MIB."
    MODULE  -- this module
        MANDATORY-GROUPS { entityPhysicalGroup,
                           entityLogicalGroup,
                           entityMappingGroup,
                           entityGeneralGroup,
                           entityNotificationsGroup }
    ::= { entityCompliances 1 }

-- MIB groupings

entityPhysicalGroup    OBJECT-GROUP
    OBJECTS {
              entPhysicalDescr,
              entPhysicalVendorType,
              entPhysicalContainedIn,
              entPhysicalClass,
              entPhysicalParentRelPos,
              entPhysicalName
            }
    STATUS  current
    DESCRIPTION
            "The collection of objects which are used to represent
            physical system components, for which a single agent
            provides management information."
    ::= { entityGroups 1 }

entityLogicalGroup    OBJECT-GROUP
    OBJECTS {
              entLogicalDescr,
              entLogicalType,
              entLogicalCommunity,
              entLogicalTAddress,
              entLogicalTDomain
            }
    STATUS  current
    DESCRIPTION
            "The collection of objects which are used to represent the
            list of logical entities for which a single agent provides
            management information."
    ::= { entityGroups 2 }

entityMappingGroup    OBJECT-GROUP
    OBJECTS {
```

```
              entLPPhysicalIndex,
              entAliasMappingIdentifier,
              entPhysicalChildIndex
            }
    STATUS  current
    DESCRIPTION
            "The collection of objects which are used to represent the
            associations between multiple logical entities, physical
            components, interfaces, and port identifiers for which a
            single agent provides management information."
    ::= { entityGroups 3 }

entityGeneralGroup    OBJECT-GROUP
    OBJECTS {
              entLastChangeTime
            }
    STATUS  current
    DESCRIPTION
            "The collection of objects which are used to represent
            general entity information for which a single agent provides
            management information."
    ::= { entityGroups 4 }

entityNotificationsGroup NOTIFICATION-GROUP
    NOTIFICATIONS { entConfigChange }
    STATUS        current
    DESCRIPTION
            "The collection of notifications used to indicate Entity MIB
            data consistency and general status information."
    ::= { entityGroups 5 }


END
```

5. Usage Examples

The following sections iterate the instance values for two example networking devices. These examples are kept simple to make them more understandable. Auxiliary components, such as fans, sensors, empty slots, and sub-modules are not shown, but might be modeled in real implementations.

5.1. Router/Bridge

A 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)
     entPhysicalParentRelPos.1 ==      0
     entPhysicalName.1 ==              '100-A'

   2 slots within the chassis:
     entPhysicalDescr.2 ==             "Acme Chassis Slot Type AA"
     entPhysicalVendorType.2  ==       acmeProducts.slotTypes.1
     entPhysicalContainedIn.2 ==       1
     entPhysicalClass.2 ==             container(5)
     entPhysicalParentRelPos.2 ==      1
     entPhysicalName.2 ==              'S1'

     entPhysicalDescr.3 ==             "Acme Chassis Slot Type AA"
     entPhysicalVendorType.3  ==       acmeProducts.slotTypes.1
     entPhysicalContainedIn.3 ==       1
     entPhysicalClass.3 ==             container(5)
     entPhysicalParentRelPos.3 ==      2
     entPhysicalName.3 ==              'S2'

   2 Field-replaceable modules:
   Slot 1 contains a module with 3 ports:
     entPhysicalDescr.4 ==             "Acme Router-100"
     entPhysicalVendorType.4  ==       acmeProducts.moduleTypes.14
     entPhysicalContainedIn.4 ==       2
     entPhysicalClass.4 ==             module(9)
     entPhysicalParentRelPos.4 ==      1
     entPhysicalName.4 ==              'M1'

     entPhysicalDescr.5 ==             "Acme Ethernet-100 Port Rev G"
```

```
   entPhysicalVendorType.5  ==       acmeProducts.portTypes.2
   entPhysicalContainedIn.5 ==       4
   entPhysicalClass.5 ==             port(10)
   entPhysicalParentRelPos.5 ==      1
   entPhysicalName.5 ==              'P1'

   entPhysicalDescr.6 ==             "Acme Ethernet-100 Port Rev G"
   entPhysicalVendorType.6  ==       acmeProducts.portTypes.2
   entPhysicalContainedIn.6 ==       4
   entPhysicalClass.6 ==             port(10)
   entPhysicalParentRelPos.6 ==      2
   entPhysicalName.6 ==              'P2'

   entPhysicalDescr.7 ==             "Acme Router-100 F-Port: Rev B"
   entPhysicalVendorType.7  ==       acmeProducts.portTypes.3
   entPhysicalContainedIn.7 ==       4
   entPhysicalClass.7 ==             port(10)
   entPhysicalParentRelPos.7 ==      3
   entPhysicalName.7 ==              'P3'

Slot 2 contains another 3-port module:
   entPhysicalDescr.8 ==             "Acme Router-100 Comm Module: Rev C"
   entPhysicalVendorType.8  ==       acmeProducts.moduleTypes.15
   entPhysicalContainedIn.8 ==       3
   entPhysicalClass.8 ==             module(9)
   entPhysicalParentRelPos.8 ==      1
   entPhysicalName.8 ==              'M2'

   entPhysicalDescr.9 ==             "Acme Fddi-100 Port Rev CC"
   entPhysicalVendorType.9 ==        acmeProducts.portTypes.5
   entPhysicalContainedIn.9 ==       8
   entPhysicalClass.9 ==             port(10)
   entPhysicalParentRelPos.9 ==      1
   entPhysicalName.9 ==              'FDDI Primary'

   entPhysicalDescr.10 ==            "Acme Ethernet-100 Port Rev G"
   entPhysicalVendorType.10 ==       acmeProducts.portTypes.2
   entPhysicalContainedIn.10 ==      8
   entPhysicalClass.10 ==            port(10)
   entPhysicalParentRelPos.10 ==     2
   entPhysicalName.10 ==             'Ethernet A'

   entPhysicalDescr.11 ==            "Acme Ethernet-100 Port Rev G"
   entPhysicalVendorType.11 ==       acmeProducts.portTypes.2
   entPhysicalContainedIn.11 ==      8
   entPhysicalClass.11 ==            port(10)
   entPhysicalParentRelPos.11 ==     3
   entPhysicalName.11 ==             'Ethernet B'
```

```
   Logical entities -- entLogicalTable
    2 OSPF instances:
      entLogicalDescr.1 ==            "Acme OSPF v1.1"
      entLogicalType.1 ==             ospf
      entLogicalCommunity.1 ==        "public-ospf1"
      entLogicalTAddress.1 ==         124.125.126.127:161
      entLogicalTDomain.1 ==          snmpUDPDomain

      entLogicalDescr.2 ==            "Acme OSPF v1.1"
      entLogicalType.2 ==             ospf
      entLogicalCommunity.2 ==        "public-ospf2"
      entLogicalTAddress.2 ==         124.125.126.127:161
      entLogicalTDomain.2 ==          snmpUDPDomain

    2 logical bridges:
      entLogicalDescr.3 ==            "Acme Bridge v2.1.1"
      entLogicalType.3  ==            dod1dBridge
      entLogicalCommunity.3 ==        "public-bridge1"
      entLogicalTAddress.3 ==         124.125.126.127:161
      entLogicalTDomain.3 ==          snmpUDPDomain

      entLogicalDescr.4 ==            "Acme Bridge v2.1.1"
      entLogicalType.4 ==             dod1dBridge
      entLogicalCommunity.4 ==        "public-bridge2"
      entLogicalTAddress.4 ==         124.125.126.127:161
      entLogicalTDomain.4 ==          snmpUDPDomain

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

  [ed. -- Note that these mappings are included in the table since
  another logical entity (1st OSPF) utilizes one of the
  ports. If this were not the case, then a single mapping
  to the module (e.g. entLPPhysicalIndex.3.4) would be
  present instead. ]
      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

Physical to Logical to MIB Alias Mappings -- entAliasMappingTable:
  Example 1: ifIndex values are global to all logical entities
      entAliasMappingIdentifier.5.0   ==        ifIndex.1
      entAliasMappingIdentifier.6.0   ==        ifIndex.2
      entAliasMappingIdentifier.7.0   ==        ifIndex.3
      entAliasMappingIdentifier.9.0   ==        ifIndex.4
      entAliasMappingIdentifier.10.0  ==        ifIndex.5
      entAliasMappingIdentifier.11.0  ==        ifIndex.6

  Example 2: ifIndex values are not shared by all logical entities
      entAliasMappingIdentifier.5.0   ==        ifIndex.1
      entAliasMappingIdentifier.5.3   ==        ifIndex.101
      entAliasMappingIdentifier.6.0   ==        ifIndex.2
      entAliasMappingIdentifier.6.3   ==        ifIndex.102
      entAliasMappingIdentifier.7.0   ==        ifIndex.3
      entAliasMappingIdentifier.7.3   ==        ifIndex.103
      entAliasMappingIdentifier.9.0   ==        ifIndex.4
      entAliasMappingIdentifier.9.3   ==        ifIndex.204
      entAliasMappingIdentifier.10.0  ==        ifIndex.5
      entAliasMappingIdentifier.10.3  ==        ifIndex.205
      entAliasMappingIdentifier.11.0  ==        ifIndex.6
      entAliasMappingIdentifier.11.3  ==        ifIndex.206

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 3 ports:
      entPhysicalChildIndex.4.5 = 5
      entPhysicalChildIndex.4.6 = 6
      entPhysicalChildIndex.4.7 = 7

  module 2 has 3 ports:
      entPhysicalChildIndex.8.9 = 9
      entPhysicalChildIndex.8.10 = 10
      entPhysicalChildIndex.1.11 = 11
```

5.2. Repeaters

A 3-slot Hub with 2 backplane ethernet segments. Slot three is empty, and the remaining slots contain ethernet repeater modules. [ed. -- Note that a replacement for the current Repeater MIB (RFC 1516) is likely to emerge soon, and it will no longer be necessary to access repeater MIB data in different naming scopes.]

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

   2 Chassis Ethernet Backplanes:
      entPhysicalDescr.2 ==          "Acme Ethernet Backplane Type A"
      entPhysicalVendorType.2 ==     acmeProducts.backplaneTypes.1
      entPhysicalContainedIn.2 ==    1
      entPhysicalClass.2 ==          backplane(4)
      entPhysicalParentRelPos.2 ==   1
      entPhysicalName.2 ==           'B1'

      entPhysicalDescr.3 ==          "Acme Ethernet Backplane Type A"
      entPhysicalVendorType.3  ==    acmeProducts.backplaneTypes.1
      entPhysicalContainedIn.3 ==    1
      entPhysicalClass.3 ==          backplane(4)
      entPhysicalParentRelPos.3 ==   2
      entPhysicalName.3 ==           'B2'

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

      entPhysicalDescr.5 ==          "Acme Hub Slot Type RB"
      entPhysicalVendorType.5  ==    acmeProducts.slotTypes.5
      entPhysicalContainedIn.5 ==    1
      entPhysicalClass.5 ==          container(5)
      entPhysicalParentRelPos.5 ==   2
      entPhysicalName.5 ==           'Slot 2'

      entPhysicalDescr.6 ==          "Acme Hub Slot Type RB"
```

```
      entPhysicalVendorType.6  ==    acmeProducts.slotTypes.5
      entPhysicalContainedIn.6 ==    1
      entPhysicalClass.6 ==          container(5)
      entPhysicalParentRelPos.6 ==   3
      entPhysicalName.6 ==           'Slot 3'

   Slot 1 contains a plug-in module with 4 10-BaseT ports:
      entPhysicalDescr.7  ==         "Acme 10Base-T Module 114 Rev A"
      entPhysicalVendorType.7   ==   acmeProducts.moduleTypes.32
      entPhysicalContainedIn.7  ==   4
      entPhysicalClass.7 ==          module(9)
      entPhysicalParentRelPos.7 ==   1
      entPhysicalName.7 ==           'M1'

      entPhysicalDescr.8  ==         "Acme 10Base-T Port RB Rev A"
      entPhysicalVendorType.8   ==   acmeProducts.portTypes.10
      entPhysicalContainedIn.8  ==   7
      entPhysicalClass.8 ==          port(10)
      entPhysicalParentRelPos.8 ==   1
      entPhysicalName.8 ==           'Ethernet-A'

      entPhysicalDescr.9  ==         "Acme 10Base-T Port RB Rev A"
      entPhysicalVendorType.9   ==   acmeProducts.portTypes.10
      entPhysicalContainedIn.9  ==   7
      entPhysicalClass.9 ==          port(10)
      entPhysicalParentRelPos.9 ==   2
      entPhysicalName.9 ==           'Ethernet-B'

      entPhysicalDescr.10 ==         "Acme 10Base-T Port RB Rev B"
      entPhysicalVendorType.10  ==   acmeProducts.portTypes.10
      entPhysicalContainedIn.10 ==   7
      entPhysicalClass.10 ==         port(10)
      entPhysicalParentRelPos.10 ==  3
      entPhysicalName.10 ==          'Ethernet-C'

      entPhysicalDescr.11 ==         "Acme 10Base-T Port RB Rev B"
      entPhysicalVendorType.11  ==   acmeProducts.portTypes.10
      entPhysicalContainedIn.11 ==   7
      entPhysicalClass.11 ==         port(10)
      entPhysicalParentRelPos.11 ==  4
      entPhysicalName.11 ==          'Ethernet-D'

   Slot 2 contains another ethernet module with 2 ports.
      entPhysicalDescr.12 ==         "Acme 10Base-T Module Model 4 Rev A"
      entPhysicalVendorType.12 ==    acmeProducts.moduleTypes.30
      entPhysicalContainedIn.12 =    5
      entPhysicalClass.12 ==         module(9)
      entPhysicalParentRelPos.12 ==  1
```

```
      entPhysicalName.12 ==          'M2'

      entPhysicalDescr.13 ==         "Acme 802.3 AUI Port Rev A"
      entPhysicalVendorType.13  ==   acmeProducts.portTypes.11
      entPhysicalContainedIn.13 ==   12
      entPhysicalClass.13 ==         port(10)
      entPhysicalParentRelPos.13 ==  1
      entPhysicalName.13 ==          'AUI'

      entPhysicalDescr.14 ==         "Acme 10Base-T Port RD Rev B"
      entPhysicalVendorType.14  ==   acmeProducts.portTypes.14
      entPhysicalContainedIn.14 ==   12
      entPhysicalClass.14 ==         port(10)
      entPhysicalParentRelPos.14 ==  2
      entPhysicalName.14 ==          'E2'

Logical entities -- entLogicalTable
   Repeater 1--comprised of any ports attached to backplane 1
      entLogicalDescr.1 ==         "Acme repeater v3.1"
      entLogicalType.1  ==         snmpDot3RptrMgt
      entLogicalCommunity.1        "public-repeater1"
      entLogicalTAddress.1 ==      124.125.126.127:161
      entLogicalTDomain.1 ==       snmpUDPDomain

   Repeater 2--comprised of any ports attached to backplane 2:
      entLogicalDescr.2 ==         "Acme repeater v3.1"
      entLogicalType.2  ==         snmpDot3RptrMgt
      entLogicalCommunity.2 ==     "public-repeater2"
      entLogicalTAddress.2 ==      124.125.126.127:161
      entLogicalTDomain.2 ==       snmpUDPDomain

Logical to Physical Mappings -- entLPMappingTable:

  repeater1 uses backplane 1, slot 1-ports 1 & 2, slot 2-port 1
  [ed. -- Note that a mapping to the module is not included,
   since in this example represents a port-switchable hub.
   Even though all ports on the module could belong to the
   same repeater as a matter of configuration, the LP port
   mappings should not be replaced dynamically with a single
   mapping for the module (e.g. entLPPhysicalIndex.1.7).
   If all ports on the module shared a single backplane connection,
   then a single mapping for the module would be more appropriate. ]

     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

Physical to Logical to MIB Alias Mappings -- entAliasMappingTable:
  Repeater Port Identifier values are shared by both repeaters:
      entAliasMappingIdentifier.8.0 ==  rptrPortGroupIndex.1.1
      entAliasMappingIdentifier.9.0 ==  rptrPortGroupIndex.1.2
      entAliasMappingIdentifier.10.0 == rptrPortGroupIndex.1.3
      entAliasMappingIdentifier.11.0 == rptrPortGroupIndex.1.4
      entAliasMappingIdentifier.13.0 == rptrPortGroupIndex.2.1
      entAliasMappingIdentifier.14.0 == 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
  [ed. - in this example, container 3 is empty.]

  module 1 has 4 ports:
      entPhysicalChildIndex.7.8 = 8
      entPhysicalChildIndex.7.9 = 9
      entPhysicalChildIndex.7.10 = 10
      entPhysicalChildIndex.7.11 = 11

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

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

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

[5] SNMPv2 Working Group, Case, J., McCloghrie, K., Rose, M., and S. Waldbusser, "Conformance Statements for version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1904, January 1996.

[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] SNMPv2 Working Group, Case, J., McCloghrie, K., Rose, M., and S. Waldbusser, "Transport Mappings for version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1906, January 1996.

[9] SNMPv2 Working Group, Case, J., McCloghrie, K., Rose, M., and S. Waldbusser, "Introduction to Community-based SNMPv2", RFC 1901, January 1996.

8. Security Considerations

In order to implement this MIB, an agent must make certain management information available about various logical and physical entities within a managed system, which may be considered sensitive in some network environments.

Therefore, a network administrator may wish to employ instance-level access control, and configure the Entity MIB access (i.e., community strings in SNMPv1 and SNMPv2C), such that certain instances within this MIB (e.g., entLogicalCommunity, or entire entLogicalEntries, entPhysicalEntries, and associated mapping table entries), are excluded from particular MIB views.

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
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134

Phone: 408-527-3711
EMail: abierman@cisco.com