Network Working Group
Request for Comments: 3512
Category: Informational

M. MacFaden
Riverstone Networks, Inc.
D. Partain
Ericsson
J. Saperia
JDS Consulting, Inc.
W. Tackabury
Gold Wire Technology, Inc.

April 2003

Configuring Networks and Devices with Simple Network Management Protocol (SNMP)

Status of this Memo

Copyright Notice

Abstract

This document is written for readers interested in the Internet Standard Management Framework and its protocol, the Simple Network Management Protocol (SNMP). In particular, it offers guidance in the effective use of SNMP for configuration management. This information is relevant to vendors that build network elements, management application developers, and those that acquire and deploy this technology in their networks.

Table of Contents

1. Introduction

1.1. The Internet Standard Management Framework

The Internet Standard Management Framework has many components. The purpose of this document is to describe effective ways of applying those components to the problems of configuration management.

For reference purposes, the Internet Standard Management Framework presently consists of five major components:

- An overall architecture, described in RFC 3411 [1].

- Mechanisms for describing and naming objects and events for the purpose of management. The first version of this Structure of Management Information (SMI) is called SMIv1 and described in STD 16, RFC 1155 [15], STD 16, RFC 1212 [16] and RFC 1215 [17]. The second version, called SMIv2, is described in STD 58, RFC 2578 [2], STD 58, RFC 2579 [3] and STD 58, RFC 2580 [4].

- Message protocols for transferring management information. The first version of the SNMP message protocol is called SNMPv1 and described in STD 15, RFC 1157 [18]. A second version of the SNMP message protocol, which is not an Internet standards track protocol, is called SNMPv2c and described in RFC 1901 [19]. The third version of the message protocol is called SNMPv3 and described in RFC 3417 [5], RFC 3412 [6] and RFC 3414 [7].

- Protocol operations for accessing management information. The first set of protocol operations and associated PDU formats is described in STD 15, RFC 1157 [18]. A second set of protocol operations and associated PDU formats is described in RFC 3416 [8].

- A set of fundamental applications described in RFC 3413 [9] and the view-based access control mechanism described in RFC 3415 [10].

A more detailed introduction to the current SNMP Management Framework can be found in RFC 3410 [12].

Managed objects are accessed via a virtual information store, termed the Management Information Base or MIB. Objects in the MIB are defined using the mechanisms defined in the SMI.

1.2. Configuration and the Internet Standard Management Framework

Data networks have grown significantly over the past decade. This growth can be seen in terms of:

Scale - Networks have more network elements, and the network elements are larger and place more demands on the systems managing them. For example, consider a typical number and speed of interfaces in a modern core network element. A managed metropolitan area network switch can have a port density much greater than the port density built into the expectations of the management systems that predated it. There are also many more interrelationships within and between devices and device functions.

Functionality - network devices perform more functions. More protocols and network layers are required for the successful deployment of network services which depend on them.

Rate of Change - the nature of modern network services causes updates, additions, and deletions of device configuration information more often than in the past. No longer can it be assumed that a configuration will be specified once and then be updated rarely. On the contrary, the trend has been towards much more frequent changes of configuration information.

Correct configuration of network elements that make up data networks is a prerequisite to the successful deployment of the services on them. The growth in size and complexity of modern networks increases the need for a standard configuration mechanism that is tightly integrated with performance and fault management systems.

The Internet Standard Management Framework has been used successfully to develop configuration management systems for a broad range of devices and networks. A standard configuration mechanism that tightly integrates with performance and fault systems is needed not only to help reduce the complexity of management, but also to enable verification of configuration activities that create revenue-producing services.

This document describes Current Practices that have been used when designing effective configuration management systems using the Internet Standard Management Framework (colloquially known as SNMP). It covers many basic practices as well as more complex agent and manager design issues that are raised by configuration management. We are not endeavoring to present a comprehensive how-to document for generalized SNMP agent, MIB module, or management application design and development. We will, however, cover points of generalized SNMP software design and implementation practice, where the practice has been seen to benefit configuration management software. So, for example, the requirement for management applications to be aware of agent limitations is discussed in the context of configuration operations, but many issues that a management application developer should consider with regard to manager-agent interactions are left for other documents and resources.

Significant experience has been gained over the past ten years in configuring public and private data networks with SNMP. During this time, networks have grown significantly as described above. A response to this explosive growth has been the development of policy-based configuration management. Policy-Based Configuration Management is a methodology wherein configuration information is derived from rules and network-wide objectives, and is distributed to potentially many network elements with the goal of achieving consistent network behavior throughout an administrative domain.

This document presents lessons learned from these experiences and applies them to both conventional and policy-based configuration systems based on SNMP.

2. Using SNMP as a Configuration Mechanism

Configuration activity causes one or more state changes in an element. While it often takes an arbitrary number of commands and amount of data to make up configuration change, it is critical that the configuration system treat the overall change operation atomically so that the number of states into which an element transitions is minimized. The goal is for a change request either to be completely executed or not at all. This is called transactional integrity. Transactional integrity makes it possible to develop

reliable configuration systems that can invoke transactions and keep track of an element's overall state and work in the presence of error states.

2.1. Transactions and SNMP

Transactions can logically take place at very fine-grained levels such as an individual object instance or in very large aggregations that could include many object instances located in many tables on a managed device. For this reason, reliance on transactional integrity only at the SNMP protocol level is insufficient.

2.2. Practical Requirements for Transactional Control

A well-designed and deployed configuration system should have the following features with regard to transactions and transactional integrity.

1) Provide for flexible transaction control at many different levels of granularity. At one extreme, an entire configuration may be delivered and installed on an element, or alternately one small attribute may be changed.

2) The transaction control component should work at and understand a notion of the kind of multi-level "defaulting" as described in Section 7.1. The key point here is that it may make most sense to configure systems at an abstract level rather than on an individual instance by instance basis as has been commonly practiced. In some cases it is more effective to send a configuration command to a system that contains a set of 'defaults' to be applied to instances that meet certain criteria.

3) An effective configuration management system must allow flexibility in the definition of a successful transaction. This cannot be done at the protocol level alone, but rather must be provided for throughout the application and the information that is being managed. In the case of SNMP, the information would be in properly defined MIB modules.

4) A configuration management system should provide time-indexed transaction control. For effective rollback control, the configuration transactions and their successful or unsuccessful completion status must be reported by the managed elements and stored in a repository that supports such time indexing and can record the user that made the change, even if the change was not carried out by the system recording the change.

5) The managed system must support transactional security. This means that depending on who is making the configuration request and where it is being made, it may be accepted or denied based on security policy that is in effect in the managed element.

Effective transactional control is a responsibility shared between design, implementation, and operational practice. Transaction control techniques for MIB module design are discussed in Section 3.3. Transaction control considerations for the agent implementation are discussed in Section 5.2.2.

2.3. Practices in Configuration--Verification

Verification of expected behavior subsequent to the commitment of change is an integral part of the configuration process. To reduce the chance of making simple errors in configuration, many organizations employ the following change management procedure:

pre-test - verify that the system is presently working properly

change - make configuration changes and wait for convergence (system or network stability)

re-test - verify once again that the system is working properly

This procedure is commonly used to verify configuration changes to critical systems such as the domain name system (DNS). DNS software kits provide diagnostic tools that allow automatic test procedures/scripts to be conducted.

A planned configuration sequence can be aborted if the pre-configuration test result shows the state of the system as unstable. Debugging the unintended effects of two sets of changes in large systems is often more challenging than an analysis of the effects of a single set after test termination.

Networks and devices under SNMP configuration readily support this change management procedure since the SNMP provides integrated monitoring, configuration and diagnostic capabilities. The key is the sequencing of SNMP protocol operations to effect an integrated change procedure like the one described above. This is usually a well-bounded affair for changes within a single network element or node. However, there are times when configuration of a given element can impact other elements in a network. Configuring network protocols such as IEEE 802.1D Spanning Tree or OSPF is especially challenging since the impact of a configuration change can directly affect stability (convergence) of the network the device is connected to.

An integrated view of configuration and monitoring provides an ideal platform from which to evaluate such changes. For example, the MIB module governing IEEE 802.1D Spanning Tree (RFC 1493 [24]) provides the following object to monitor stability per logical bridge.

```
dot1dStpTopChanges OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
       "The total number of topology changes detected by
       this bridge since the management entity was last
       reset or initialized."
    REFERENCE
       "IEEE 802.1D-1990: Section 6.8.1.1.3"
    ::= { dot1dStp 4 }
```

Likewise, the OSPF MIB module provides a similar metric for stability per OSPF area.

```
ospfSpfRuns OBJECT-TYPE
    SYNTAX   Counter32
    MAX-ACCESS   read-only
    STATUS   current
    DESCRIPTION
       "The number of times that the intra-area route
       table has been calculated using this area's
       link-state database.  This is typically done
       using Dijkstra's algorithm."
   ::= { ospfAreaEntry 4 }
```

The above object types are good examples of a means of facilitating the principles described in Section 2.3. That is, one needs to understand the behavior of a subsystem before configuration change, then be able to use the same means to retest and verify proper operation subsequent to configuration change.

The operational effects of a given implementation often differ from one to another for any given standard configuration object. The impact of a change to stability of systems such as OSPF should be documented in an agent-capabilities statement which is consistent with "Requirements for IP Version 4 Routers" [22], Section 1.3.4:

> A vendor needs to provide adequate documentation on all configuration parameters, their limits and effects.

Adherence to the above model is not fail-safe, especially when configuration errors are masked by long latencies or when configuration errors lead to oscillations in network stability. For example, consider the situation of loading a new software version on a device, which leads to small, slow, cumulative memory leaks brought on by a certain traffic pattern that was not caught during vendor and customer test lab trials.

In a network-based example, convergence in an autonomous system cannot be guaranteed when configuration changes are made since there are factors beyond the control of the operator, such as the state of other network elements. Problems affecting this convergence may not be detected for a significant period of time after the configuration change. Even for factors within the operator's control, there is often little verification done to prevent mis-configuration (as shown in the following example).

Consider a change made to ospfIfHelloInterval and ospfIfRtrDeadInterval [24] timers in the OSPF routing protocol such that both are set to the same value. Two routers may form an adjacency but then begin to cycle in and out of adjacency, and thus never reach a stable (converged) state. Had the configuration process described at the beginning of this section been employed, this particular situation would have been discovered without impacting the production network.

The important point to remember from this discussion is that configuration systems should be designed and implemented with verification tests in mind.

3. Designing a MIB Module

Carefully considered MIB module designs are crucial to practical configuration with SNMP. As we have just seen, MIB objects designed for configuration can be very effective since they can be associated with integrated diagnostic, monitoring, and fault objects. MIB modules for configuration also scale when they expose their notion of template object types. Template objects can represent information at a higher level of abstraction than instance-level ones. This has the benefit of reducing the amount of instance-level data to move from management application to the agent on the managed element, when that instance-level data is brought about by applying a template object on the agent. Taken together, all of these objects can provide a robust configuration subsystem.

The remainder of this section provides specific practices used in MIB module design with SMIv2 and SNMPv3.

3.1. MIB Module Design - General Issues

One of the first tasks in defining a MIB module is the creation of a model that reflects the scope and organization of the management information an agent will expose.

MIB modules can be thought of as logical models providing one or more aspects/views of a subsystem. The objective for all MIB modules should be to serve one or more operational requirements such as accounting information collection, configuration of one or more parts of a system, or fault identification. However, it is important to include only those aspects of a subsystem that are proven to be operationally useful.

In 1993, one of most widely deployed MIB modules supporting configuration was published, RFC 1493, which contained the BRIDGE-MIB. It defined the criteria used to develop the MIB module as follows:

> To be consistent with IAB directives and good engineering practice, an explicit attempt was made to keep this MIB as simple as possible. This was accomplished by applying the following criteria to objects proposed for inclusion:

(1) Start with a small set of essential objects and add only as further objects are needed.

(2) Require objects be essential for either fault or configuration management.

(3) Consider evidence of current use and/or utility.

(4) Limit the total (sic) of objects.

(5) Exclude objects which are simply derivable from others in this or other MIBs.

(6) Avoid causing critical sections to be heavily instrumented. The guideline that was followed is one counter per critical section per layer.

Over the past eight years additional experience has shown a need to expand these criteria as follows:

(7) Before designing a MIB module, identify goals and objectives for the MIB module. How much of the underlying system will be exposed depends on these goals.

(8) Minimizing the total number of objects is not an explicit goal, but usability is. Be sure to consider deployment and usability requirements.

(9) During configuration, consider supporting explicit error state, capability and capacity objects.

(10) When evaluating rule (5) above, consider the impact on a management application. If an object can help reduce a management application's complexity, consider defining objects that can be derived.

3.2. Naming MIB modules and Managed Objects

Naming of MIB modules and objects informally follows a set of best practices. Originally, standards track MIB modules used RFC names. As the MIB modules evolved, the practice changed to using more descriptive names. Presently, Standards Track MIB modules define a given area of technology such as ATM-MIB, and vendors then extend such MIB modules by prefixing the company name to a given MIB module as in ACME-ATM-MIB.

Object descriptors (the "human readable names" assigned to object identifiers [2]) defined in standard MIB modules should be unique across all MIB modules. Generally, a prefix is added to each managed object that can help reference the MIB module it was defined in. For example, the IF-MIB uses "if" prefix for descriptors of object types such as ifTable, ifStackTable and so forth.

MIB module object type descriptors can include an abbreviation for the function they perform. For example the objects that control configuration in the example MIB module in Section 8 include "Cfg" as part of the object descriptor, as in bldgHVACCfgDesiredTemp.

This is more fully realized when the object descriptors that include the fault, configuration, accounting, performance and security [33] abbreviations are combined with an organized OID assignment approach. For example, a vendor could create a configuration branch in their private enterprises area. In some cases this might be best done on a per product basis. Whatever the approach used, "Cfg" might be included in every object descriptor in the configuration branch. This has two operational benefits. First, for those that do look at instances of MIB objects, descriptors as seen through MIB browsers or other command line tools assist in conveying the meaning of the object type. Secondly, management applications can be pointed at specific subtrees for fault or configuration, causing a more efficient retrieval of data and a simpler management application with potentially better performance.

3.3. Transaction Control And State Tracking

Transactions and keeping track of their state is an important consideration when performing any type of configuration activity regardless of the protocol. Here are a few areas to consider when designing transaction support into an SNMP-based configuration system.

3.3.1. Conceptual Table Row Modification Practices

Any discussion of transaction control as it pertains to MIB module design often begins with how the creation or modification of object instances in a conceptual row in the MIB module is controlled.

RowStatus [3] is a standard textual convention for the management of conceptual rows in a table. Specifically, the RowStatus textual convention that is used for the SYNTAX value of a single column in a table controls the creation, deletion, activation, and deactivation of conceptual rows of the table. When a table has been defined with a RowStatus object as one of its columns, changing an instance of the object to 'active' causes the row in which that object instance appears to become 'committed'.

In a multi-table scenario where the configuration data must be spread over many columnar objects, a RowStatus object in one table can be used to cause the entire set of data to be put in operation or stored based on the definition of the objects.

In some cases, very large amounts of data may need to be 'committed' all at once. In these cases, another approach is to configure all of the rows in all the tables required and have an "activate" object that has a set method that commits all the modified rows.

The RowStatus textual convention specifies that, when used in a conceptual row, a description must define what can be modified. While the description of the conceptual row and its columnar object types is the correct place to derive this information on instance modifiability, it is often wrongly assumed in some implementations that:

1) objects either must all be presently set or none need be set to make a conceptual RowStatus object transition to active(1)

2) objects in a conceptual row cannot be modified once a RowStatus object is active(1). Restricting instance modifiability like this, so that after a RowStatus object is set to active(1) is in fact a reasonable limitation, since such a set of RowStatus may have agent system side-effects which depend on committed columnar

object instance values. However, where this restriction exists on an object, it should be made clear in a DESCRIPTION clause such as the following:

```
protocolDirDescr OBJECT-TYPE
  SYNTAX      DisplayString (SIZE (1..64))
  MAX-ACCESS  read-create
  STATUS      current
  DESCRIPTION
      "A textual description of the protocol encapsulation.
      A probe may choose to describe only a subset of the
      entire encapsulation (e.g., only the highest layer).

      This object is intended for human consumption only.

      This object may not be modified if the associated
      protocolDirStatus object is equal to active(1)."
  ::= { protocolDirEntry 4 }
```

Any such restrictions on columnar object instance modification while a row's RowStatus object instance is set to active(1) should appear in the DESCRIPTION clause of the RowStatus columnar OBJECT-TYPE as well.

3.3.2. Fate sharing with multiple tables

An important principle associated with transaction control is fate sharing of rows in different tables. Consider the case where a relationship has been specified between two conceptual tables of a MIB module (or tables in two different MIB modules). In this context, fate sharing means that when a row of a table is deleted, the corresponding row in the other table is also deleted. Fate sharing in a transaction control context can also be used with the activation of very large configuration changes. If we have two tables that hold a set of configuration information, a row in one table might have to be put in the 'ready' state before the second can be put in the 'ready' state. When that second table can be placed in the 'ready' state, then the entire transaction can be considered to have been 'committed'.

Fate sharing of SNMP table data should be explicitly defined where possible using the SMI index qualifier AUGMENTS. If the relationship between tables cannot be defined using SMIv2 macros, then the DESCRIPTION clause of the object types which particularly effect the cross-table relationship should define what should happen when rows in related tables are added or deleted.

Consider the relationship between the dot1dBasePortTable and the ifTable. These tables have a sparse relationship. If a given ifEntry supports 802.1D bridging then there is a dot1dBasePortEntry that has a pointer to it via dot1dBasePortIfIndex.

Now, what should happen if an ifEntry that can bridge is deleted? Should the object dot1dBasePortIfIndex simply be set to 0 or should the dot1dBasePortEntry be deleted as well? A number of acceptable design and practice techniques can provide the answer to these questions, so it is important for the MIB module designer to provide the guidance to guarantee consistency and interoperability.

To this end, when two tables are related in such a way, ambiguities such as this should be avoided by having the DESCRIPTION clauses of the pertinent row object types define the fate sharing of entries in the respective tables.

3.3.3. Transaction Control MIB Objects

When a MIB module is defined that includes configuration object types, consider providing transaction control objects. These objects can be used to cause a large transaction to be committed. For example, we might have several tables that define the configuration of a portion of a system. In order to avoid churn in the operational state of the system we might create a single scalar object that, when set to a particular value, will cause the activation of the rows in all the necessary tables. Here are some examples of further usage for such object types:

- Control objects that are the 'write' or 'commit' objects.

  Such objects can cause all pending transactions (change MIB object values as a result of SET operations) to be committed to a permanent repository or operational memory, as defined by the semantics of the MIB objects.

- Control objects at different levels of configuration granularity.

  One of the decisions for a MIB module designer is what are the levels of granularity that make sense in practice. For example, in the routing area, would changes be allowed on a per protocol basis such as BGP? If allowed at the BGP level, are sub-levels permitted such as per autonomous system? The design of these control objects will be impacted by the underlying software design. RowStatus (see Section 3.3.1) also has important relevance as a general transaction control object.

3.3.4. Creating And Activating New Table Rows

When designing read-create objects in a table, a MIB module designer should first consider the default state of each object in the table when a row is created. Should an implementation of a standard MIB module vary in terms of the objects that need to be set in order to create an instance of a given row, an agent capabilities statement should be used to name the additional objects in that table using the CREATION-REQUIRES clause.

It is useful when configuring new rows to use the notReady status to indicate row activation cannot proceed.

When creating a row instance of a conceptual table, one should consider the state of instances of required columnar objects in the row. The DESCRIPTION clause of such a required columnar object should specify it as such.

During the period of time when a management application is attempting to create a row, there may be a period of time when not all of these required (and non-defaultable) columnar object instances have been set. Throughout this time, an agent should return a noSuchInstance error for a GET of any object instance of the row until such time that all of these required instance values are set. The exception is the RowStatus object instance, for which a notReady(3) value should be returned during this period.

One need only be concerned with the notReady value return for a RowStatus object when the row under creation does not yet have all of the required, non-defaultable instance values for the row. One approach to simplifying in-row configuration transactions when designing MIB modules is to construct table rows that have no more instance data for columnar objects than will fit inside a single SET PDU. In this case, the createAndWait() value for the RowStatus columnar object is not required. It is possible to use createAndGo() in the same SET PDU, thus simplifying transactional management.

3.3.5. Summary Objects and State Tracking

Before beginning a new set of configuration transactions, a management application might want to checkpoint the state of the managed devices whose configuration it is about to change. There are a number of techniques that a MIB module designer can provide to assist in the (re-)synchronization of the managed systems. These objects can also be used to verify that the management application's notion of the managed system state is the same as that of the managed device.

These techniques include:

1. Provide an object that reports the number of rows in a table

2. Provide an object that flags when data in the table was last modified.

3. Send a notification message (InformRequests are preferable) to deliver configuration change.

By providing an object containing the number of rows in a table, management applications can decide how best to retrieve a given table's data and may choose different retrieval strategies depending on table size. Note that the availability of and application monitoring of such an object is not sufficient for determining the presence of table data change over a checkpointed duration since an equal number of row creates and deletes over that duration would reflect no change in the object instance value. Additionally, table data change which does not change the number of rows in the table would not be reflected through simple monitoring of such an object instance.

Instead, the change in the value of any table object instance data can be tracked through an object that monitors table change state as a function of time. An example is found in RFC 2790, Host Resources MIB:

```
hrSWInstalledLastUpdateTime OBJECT-TYPE
    SYNTAX     TimeTicks
    MAX-ACCESS read-only
    STATUS     current
    DESCRIPTION
        "The value of sysUpTime when the hrSWInstalledTable
        was last completely updated.  Because caching of this
        data will be a popular implementation strategy,
        retrieval of this object allows a management station
        to obtain a guarantee that no data in this table is
        older than the indicated time."
    ::= { hrSWInstalled 2 }
```

A similar convention found in many standards track MIB modules is the "LastChange" type object.

For example, the ENTITY-MIB, RFC 2737 [34], provides the following object:

```
entLastChangeTime OBJECT-TYPE
  SYNTAX      TimeStamp
```

```
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
            "The value of sysUpTime at the time a conceptual row is
            created, modified, or deleted in any of these tables:
                    - entPhysicalTable
                    - entLogicalTable
                    - entLPMappingTable
                    - entAliasMappingTable
                    - entPhysicalContainsTable"
    ::= { entityGeneral 1 }
```

This convention is not formalized. There tend to be small differences in what a table's LastChanged object reflects. IF-MIB (RFC 2863 [20]) defines the following:

```
ifTableLastChange  OBJECT-TYPE
    SYNTAX      TimeTicks
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
            "The value of sysUpTime at the time of the last
            creation or deletion of an entry in the ifTable.  If
            the number of entries has been unchanged since the
            last re-initialization of the local network management
            subsystem, then this object contains a zero value."
    ::= { ifMIBObjects 5 }
```

So, if an agent modifies a row with an SNMP SET on ifAdminStatus, the value of ifTableLastChange will not be updated. It is important to be specific about what can cause an object to update so that management applications will be able to detect and more properly act on these changes.

The final way to keep distributed configuration data consistent is to use an event-driven model, where configuration changes are communicated as they occur. When the frequency of change to configuration is relatively low or polling a cache object is not desired, consider defining a notification that can be used to report all configuration change details.

When doing so, the option is available to an SNMPv3 (or SNMPv2c) agent to deliver the notification using either a trap or an inform. The decision as to which PDU to deliver to the recipient is generally a matter of local configuration. Vendors should recommend the use of informs over traps for NOTIFICATION-TYPE data since the agent can use the presence or absence of a response to help know whether it needs

to retransmit or not. Overall, it is preferable to use an inform instead of a trap so that changes have a higher likelihood of confirmed end-to-end delivery.

As a matter of MIB module design, when practical, the NOTIFICATION-TYPE should include in the PDU all of the modified columnar objects in a row of a table. This makes it easier for the management application receiving the notification to keep track of what has changed in the row of a table and perform addition analysis on the state of the managed elements.

However, the use of notifications to communicate the state of a rapidly changing object may not be ideal either. This leads us back to the MIB module design question of what is the right level of granularity to expose.

Finally, having to poll many "LastChange" objects does not scale well. Consider providing a global LastChange type object to represent overall configuration in a given agent implementation.

3.3.6. Optimizing Configuration Data Transfer

Configuration management software should keep track of the current configuration of all devices under its control. It should ensure that the result is a consistent view of the configuration of the network, which can help reduce inadvertent configuration errors.

In devices that have very large amounts of configuration data, it can be costly to both the agent and the manager to have the manager periodically poll the entire contents of these configuration tables for synchronization purposes. A benefit of good synchronization between the manager and the agent is that the manager can determine the smallest and most effective set of data to send to managed devices when configuration changes are required. Depending on the table organization in the managed device and the agent implementation, this practice can reduce the burden on the managed device for activation of these configuration changes.

In the previous section, we discussed the "LastChange" style of object. When viewed against the requirements just described, the LastChange object is insufficient for large amounts of data.

There are three design options that can be used to assist with the synchronization of the configuration data found in the managed device with the manager:

1) Design multiple indices to partition the data in a table logically or break a table into a set of tables to partition the data based on what an application will use the table for

2) Use a time-based indexing technique

3) Define a control MIB module that manages a separate data delivery protocol

3.3.6.1. Index Design

Index design has a major impact on the amount of data that must be transferred between SNMP entities and can help to mitigate scaling issues with large tables.

Many tables in standard MIB modules follow one of two indexing models:

- Indexing based upon increasing Integer32 or Unsigned32 values of the kind one might find in an array.

- Associative indexing, which refers to the technique of using potentially sparse indices based upon a "key" of the sort one would use for a hash table.

When tables grow to a very large number of rows, using an associative indexing scheme offers the useful ability to efficiently retrieve only the rows of interest.

For example, if an SNMP entity exposes a copy of the default-free Internet routing table as defined in the ipCidrRouteTable, it will presently contain around 100,000 rows.

Associative indexing is used in the ipCidrRouteTable and allows one to retrieve, for example, all routes for a given IPv4 destination 192.0.2/24.

Yet, if the goal is to extract a copy of the table, the associative indexing reduces the throughput and potentially the performance of retrieval. This is because each of the index objects are appended to the object identifiers for every object instance returned.

```
ipCidrRouteEntry OBJECT-TYPE
   SYNTAX   IpCidrRouteEntry
   MAX-ACCESS not-accessible
   STATUS   current
   DESCRIPTION
    "A particular route to a particular destination,
```

```
       under a particular policy."
      INDEX {
        ipCidrRouteDest,
        ipCidrRouteMask,
        ipCidrRouteTos,
        ipCidrRouteNextHop
        }
```

A simple array-like index works efficiently since it minimizes the index size and complexity while increasing the number of rows that can be sent in a PDU. If the indexing is not sparse, concurrency can be gained by sending multiple asynchronous non-overlapping collection requests as is explained in RFC 2819 [32], Page 41 (in the section pertaining to Host Group indexing).

> Should requirements dictate new methods of access, multiple indices can be defined such that both associative and simple indexing can coexist to access a single logical table.

Two examples follow.

First, consider the ifStackTable found in RFC 2863 [20] and the ifInvStackTable RFC 2864 [33]. They are logical equivalents with the order of the auxiliary (index) objects simply reversed.

```
ifStackEntry  OBJECT-TYPE
     SYNTAX        IfStackEntry
     MAX-ACCESS    not-accessible
     STATUS        current
     DESCRIPTION
             "Information on a particular relationship between
             two sub-layers, specifying that one sub-layer runs
             on 'top' of the other sub-layer.  Each sub-layer
             corresponds to a conceptual row in the ifTable."
             INDEX { ifStackHigherLayer, ifStackLowerLayer }
     ::= { ifStackTable 1 }

ifInvStackEntry  OBJECT-TYPE
   SYNTAX        IfInvStackEntry
   MAX-ACCESS    not-accessible
   STATUS        current
   DESCRIPTION
       "Information on a particular relationship between two
       sub-layers, specifying that one sub-layer runs underneath
       the other sub-layer.  Each sub-layer corresponds to a
       conceptual row in the ifTable."
       INDEX { ifStackLowerLayer, ifStackHigherLayer }
   ::= { ifInvStackTable 1 }
```

Second, table designs that can factor data into multiple tables with well-defined relationships can help reduce overall data transfer requirements. The RMON-MIB, RFC 2819 [32], demonstrates a very useful technique of organizing tables into control and data components. Control tables contain those objects that are configured and change infrequently, and the data tables contain information to be collected that can be large and may change quite frequently.

As an example, the RMON hostControlTable provides a way to specify how to collect MAC addresses learned as a source or destination from a given port that provides transparent bridging of Ethernet packets.

Configuration is accomplished using the hostControlTable. It is indexed by a simple integer. While this may seem to be array-like, it is common practice for command generators to encode the ifIndex into this simple integer to provide associative lookup capability.

The RMON hostTable and hostTimeTable represent dependent tables that contain the results indexed by the hostControlTable entry.

The hostTable is further indexed by the MAC address which provides the ability to reasonably search for a collection, such as the Organizationally Unique Identifier (OUI), the first three octets of the MAC address.

The hostTimeTable is designed explicitly for fast transfer of bulk RMON data. It demonstrates how to handle collecting large number of rows in the face of deletions and insertions by providing hostControlLastDeleteTime.

```
hostControlLastDeleteTime OBJECT-TYPE
SYNTAX     TimeTicks
MAX-ACCESS read-only
STATUS     current
DESCRIPTION
    "The value of sysUpTime when the last entry
    was deleted from the portion of the hostTable
    associated with this hostControlEntry.  If no
    deletions have occurred, this value shall be zero."
::= { hostControlEntry 4 }
```

3.3.6.2. Time Based Indexing

The TimeFilter as defined in RFC 2021 [44] and used in RMON2-MIB and Q-BRIDGE-MIB (RFC 2674 [26]) provides a way to obtain only those rows that have changed on or after some specified period of time has passed.

One drawback to TimeFilter index tables is that a given row can appear at many points in time, which artificially inflates the size of the table when performing standard getNext or getBulk data retrieval.

3.3.6.3. Alternate Data Delivery Mechanisms

If the amount of data to transfer is larger than current SNMP design restrictions permit, as in the case of OCTET STRINGS (64k minus overhead of IP/UDP header plus SNMP header plus varbind list plus varbind encoding), consider delivery of the data via an alternate method, such as FTP and use a MIB module to control that data delivery process. In many cases, this problem can be avoided via effective MIB design. In other words, object types requiring this kind of transfer size should be used judiciously, if at all.

There are many enterprise MIB modules that provide control of the TFTP or FTP protocol. Often the SNMP part defines what to send where and setting an object initiates the operation (for an example, refer to the CISCO-FTP-CLIENT-MIB, discussed in [38]).

Various approaches exist for allowing a local agent process running within the managed node to take a template for an object instance (for example for a set of interfaces), and adapt and apply it to all of the actual instances within the node. This is an architecture for one form of policy-based configuration (see [36], for example). Such an architecture, which must be designed into the agent and some portions of the MIB module, affords the efficiency of specifying many copies of instance data only once, along with the execution efficiency of distributing the application of the instance data to the agent.

Other work is currently underway to improve efficiency for bulk SNMP transfer operations [37]. The objective of these efforts is simply the conveyance of more information with less overhead.

3.4. More Index Design Issues

Section 3.3.5 described considerations for table row index design as it pertains to the synchronization of changes within sizable table rows. This section simply considers how to specify this syntactically and how to manage indices semantically.

In many respects, the design issues associated with indices in a MIB module are similar to those in a database. Care must be taken during the design phase to determine how often and what kind of information must be set or retrieved. The next few points provide some guidance.

3.4.1. Simple Integer Indexing

When indexing tables using simple Integer32 or Unsigned32, start with one (1) and specify the maximum range of the value. Since object identifiers are unsigned long values, a question that arises is why not index from zero (0) instead of one(1)?

RFC 2578 [2], Section 7.7, page 28 states the following: Instances identified by use of integer-valued objects should be numbered starting from one (i.e., not from zero). The use of zero as a value for an integer-valued index object type should be avoided, except in special cases. Consider the provisions afforded by the following textual convention from the Interfaces Group MIB module [33]:

```
InterfaceIndexOrZero ::= TEXTUAL-CONVENTION
    DISPLAY-HINT "d"
    STATUS       current
    DESCRIPTION
        "This textual convention is an extension of the
        InterfaceIndex convention.  The latter defines a greater
        than zero value used to identify an interface or interface
        sub-layer in the managed system.  This extension permits the
        additional value of zero.  the value zero is object-specific
        and must therefore be defined as part of the description of
        any object which uses this syntax.  Examples of the usage of
        zero might include situations where interface was unknown,
        or when none or all interfaces need to be referenced."
    SYNTAX       Integer32 (0..2147483647)
```

3.4.2. Indexing with Network Addresses

There are many objects that use IPv4 addresses (SYNTAX IpAddress) as indexes. One such table is the ipAddrTable from RFC 2011 [14] IP-MIB. This limits the usefulness of the MIB module to IPv4. To avoid such limitations, use the addressing textual conventions INET-ADDRESS-MIB [13] (or updates to that MIB module), which provides a generic way to represent addresses for Internet Protocols. In using the InetAddress textual convention in this MIB, however, pay heed to the following advisory found in its description clause:

> When this textual convention is used as the syntax of an index object, there may be issues with the limit of 128 sub-identifiers specified in SMIv2, STD 58. In this case, the OBJECT-TYPE declaration MUST include a 'SIZE' clause to limit the number of potential instance sub-identifiers.

One should consider the SMI limitation on the 128 sub-identifier specification when using certain kinds of network address index types. The most likely practical liability encountered in practice has been with DNS names, which can in fact be in excess of 128 bytes. The problem can be, of course, compounded when multiple indices of this type are specified for a table.

3.5. Conflicting Controls

MIB module designers should avoid specifying read-write objects that overlap in function partly or completely.

Consider the following situation where two read-write objects partially overlap when a dot1dBasePortEntry has a corresponding ifEntry.

The BRIDGE-MIB defines the following managed object:

```
dot1dStpPortEnable OBJECT-TYPE
    SYNTAX  INTEGER {
                enabled(1),
                disabled(2)           }
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
        "The enabled/disabled status of the port."
    REFERENCE
        "IEEE 802.1D-1990: Section 4.5.5.2"
    ::= { dot1dStpPortEntry 4 }
```

The IF-MIB defines a similar managed object:

```
ifAdminStatus OBJECT-TYPE
    SYNTAX  INTEGER {
                up(1),       -- ready to pass packets
                down(2),
                testing(3)   -- in some test mode
            }
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
        "The desired state of the interface.  The testing(3)
        state indicates that no operational packets can be
        passed.  When a managed system initializes, all
        interfaces start with ifAdminStatus in the down(2) state.
        As a result of either explicit management action or per
        configuration information retained by the managed system,
        ifAdminStatus is then changed to either the up(1) or
```

```
            testing(3) states (or remains in the down(2) state)."
        ::= { ifEntry 7 }
```

If ifAdminStatus is set to testing(3), the value to be returned for dot1dStpPortEnable is not defined. Without clarification on how these two objects interact, management implementations will have to monitor both objects if bridging is detected and correlate behavior.

The dot1dStpPortEnable object type could have been written with more information about the behavior of this object when values of ifAdminStatus which impact it change. For example, text could be added that described proper return values for the dot1dStpPortEnable object instance for each of the possible values of ifAdminStatus.

In those cases where overlap between objects is unavoidable, then as we have just described, care should be taken in the description of each of the objects to describe their possible interactions. In the case of an object type defined after an incumbent object type, it is necessary to include in the DESCRIPTION of this later object type the details of these interactions.

3.6. Textual Convention Usage

Textual conventions should be used whenever possible to create a consistent semantic for an oft-recurring datatype.

MIB modules often define a binary state object such as enable/disable or on/off. Current practice is to use existing Textual Conventions and define the read-write object in terms of a TruthValue from SNMPv2-TC [3]. For example, the Q-BRIDGE-MIB [26] defines:

```
dot1dTrafficClassesEnabled OBJECT-TYPE
    SYNTAX      TruthValue
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
       "The value true(1) indicates that Traffic Classes are
       enabled on this bridge.  When false(2), the bridge
       operates with a single priority level for all traffic."
    DEFVAL      { true }
    ::= { dot1dExtBase 2 }
```

Textual conventions that have a reasonable chance of being reused in other MIB modules ideally should also be defined in a separate MIB module to facilitate sharing of such object types. For example, all ATM MIB modules draw on the ATM-TC-MIB [39] to reference and utilize common definitions for addressing, service class values, and the like.

To simplify management, it is recommended that existing SNMPv2-TC based definitions be used when possible. For example, consider the following object definition:

```
acmePatioLights OBJECT-TYPE
    SYNTAX  INTEGER {
                on(1),
                off(2),
    }
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
        "Current status of outdoor lighting."
    ::= { acmeOutDoorElectricalEntry 3 }
```

This could be defined as follows using existing SNMPv2-TC TruthValue.

```
acmePatioLightsOn OBJECT-TYPE
    SYNTAX  TruthValue
    MAX-ACCESS  read-write
    STATUS      current
    DESCRI2096PTION
        "Current status of outdoor lighting.  When set to true (1),
        this means that the lights are enabled and turned on.
        When set to false (2), the lights are turned off."
     ::= { acmeOutDoorElectricalEntry 3 }
```

3.7. Persistent Configuration

Many network devices have two levels of persistence with regard to configuration data. In the first case, the configuration data sent to the device is persistent only until changed with a subsequent configuration operation, or the system is reinitialized. The second level is where the data is made persistent as an inherent part of the acceptance of the configuration information. Some configuration shares both these properties, that is, that on acceptance of new configuration data it is saved permanently and in memory. Neither of these necessarily means that the data is used by the operational code. Sometimes separate objects are required to activate this new configuration data for use by the operational code.

However, many SNMP agents presently implement simple persistence models, which do not reflect all the relationships of the configuration data to the actual persistence model as described above. Some SNMP set requests against MIB objects with MAX-ACCESS read-write are written automatically to a persistent store. In other cases, they are not. In some of the latter cases, enterprise MIB

objects are required in order to get standard configuration stored, thus making it difficult for a generic application to have a consistent effect.

There are standard conventions for saving configuration data. The first method uses the Textual Convention known as StorageType [3] which explicitly defines a given row's persistence requirement.

Examples include the RFC 3231 [25] definition for the schedTable row object schedStorageType of syntax StorageType, as well as similar row objects for virtually all of the tables of the SNMP View-based Access Control Model MIB [10].

A second method for persistence simply uses the DESCRIPTION clause to define how instance data should persist. RFC 2674 [26] explicitly defines Dot1qVlanStaticEntry data persistence as follows:

```
dot1qVlanStaticTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF Dot1qVlanStaticEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
     "A table containing static configuration information for
     each VLAN configured into the device by (local or
     network) management.  All entries are permanent and will
     be restored after the device is reset."
    ::= { dot1qVlan 3 }
```

The current practice is a dual persistence model where one can make changes to run-time configuration as well as to a non-volatile configuration read at device initialization. The DISMAN-SCHEDULE-MIB module [25] provides an example of this practice. A row entry of its SchedTable specifies the parameters by which an agent MIB variable instance can be set to a specific value at some point in time and governed by other constraints and directives. One of those is:

```
schedStorageType OBJECT-TYPE
     SYNTAX      StorageType
     MAX-ACCESS  read-create
     STATUS      current
     DESCRIPTION
         "This object defines whether this scheduled action is kept
          in volatile storage and lost upon reboot or if this row is
          backed up by non-volatile or permanent storage.
          Conceptual rows having the value 'permanent' must allow
          write access to the columnar objects schedDescr,
          schedInterval, schedContextName, schedVariable, schedValue,
          and schedAdminStatus.  If an implementation supports the
```

```
            schedCalendarGroup, write access must be also allowed to
            the columnar objects schedWeekDay, schedMonth, schedDay,
            schedHour, schedMinute."
        DEFVAL { volatile }
        ::= { schedEntry 19 }
```

It is important, however, to reiterate that the persistence is ultimately controlled by the capabilities and features (with respect to the storage model of management data) of the underlying system on which the MIB Module agent is being implemented. This falls into very much the same kind of issue set as, for example, the situation where the size of data storage in the system for a Counter object type is not the same as that in the corresponding MIB Object Type. To generalize, the final word on the "when" and "how" of storage of persistent data is dictated by the system and the implementor of the agent on the system.

3.8. Configuration Sets and Activation

An essential notion for configuration of network elements with SNMP is awareness of the difference between the set of one or more configuration objects from the activation of those configuration changes in the actual subsystem. That is, it often only makes sense to activate a group of objects as a single ’transaction’.

3.8.1. Operational Activation Considerations

A MIB module design must consider the implications of the preceding in the context of changes that will occur throughout a subsystem when changes are activated. This is particularly true for configuration changes that are complex. This complexity can be in terms of configuration data or the operational ramifications of the activation of the changes in the managed subsystem. A practical technique to accommodate this kind of activation is the partitioning of contained configuration sets, as it pertains to their being activated as changes. Any complex configuration should have a master on/off switch (MIB object type) as well as strategically placed on/off switches that partition the activation of configuration data in the managed subsystem. These controls play a pivotal role during the configuration process as well as during subsequent diagnostics. Generally, a series of set operations should not cause an agent to activate each object, causing operational instability to be introduced with every changed object instance. To avoid this liability, ideally a series of Set PDUs can install the configuration and a final set series of PDUs can activate the changes.

During diagnostic situations, certain on/off switches can be set to localize the perceived error instead of having to remove the configuration.

An example of such an object from the OSPF Version 2 MIB [29] is the global ospfAdminStat:

```
ospfAdminStat OBJECT-TYPE
    SYNTAX   Status
    MAX-ACCESS   read-write
    STATUS   current
    DESCRIPTION
       "The administrative status of  OSPF  in the
       router.  The value 'enabled' denotes that the
       OSPF Process is active on at least one interface;
       'disabled' disables it on all interfaces."
   ::= { ospfGeneralGroup 2 }
```

Elsewhere in the OSPF MIB, the semantics of setting ospfAdminStat to enabled(2) are clearly spelled out.

The Scheduling MIB [25] exposes such an object on each entry in the scheduled actions table, along with the corresponding stats object type (with read-only ACCESS) on the scheduled actions row instance.

This reflects a recurring basic design pattern which brings about semantic clarity in the object type usage. A table can expose one columnar object type which is strictly for administrative control. When read, an instance of this object type will reflect its last set or defaulted value. A companion operational columnar object type, with MAX-ACCESS of read-only, provides the current state of activation or deactivation resulting from the last set of the administrative columnar instance. It is fully expected that these administrative and operational columnar instances may reflect different values over some period of time of activation latency, which is why they are separate. Further sections display some of the problems which can result from attempting to combine the operational and administrative row columns into a single object type.

Note that all of this is independent of the RowStatus columnar object, and the notion of 'activation' as it pertains to RowStatus. A defined RowStatus object type should be strictly concerned with the management of the table row itself (with 'activation' indicating "the conceptual row is available for use by the managed device" [3], and not to be confused with any operational activation semantics).

In the following example, schedAdminStatus controls activation of the scheduled action, and schedOperStatus reports on its operational status:

```
schedAdminStatus OBJECT-TYPE
    SYNTAX      INTEGER {
                    enabled(1),
                    disabled(2)
                }
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "The desired state of the schedule."
    DEFVAL { disabled }
    ::= { schedEntry 14 }

schedOperStatus OBJECT-TYPE
    SYNTAX      INTEGER {
                    enabled(1),
                    disabled(2),
                    finished(3)
                }
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The current operational state of this schedule.  The state
         enabled(1) indicates this entry is active and that the
         scheduler will invoke actions at appropriate times.  The
         disabled(2) state indicates that this entry is currently
         inactive and ignored by the scheduler.  The finished(3)
         state indicates that the schedule has ended.  Schedules
         in the finished(3) state are ignored by the scheduler.
         A one-shot schedule enters the finished(3) state when it
         deactivates itself."
    ::= { schedEntry 15 }
```

3.8.2. RowStatus and Deactivation

RowStatus objects should not be used to control activation/deactivation of a configuration. While RowStatus looks ideally suited for such a purpose since a management application can set a row to active(1), then set it to notInService(2) to disable it then make it active(1) again, there is no guarantee that the agent won't discard the row while it is in the notInService(2) state. RFC 2579 [3], page 15 states:

The agent must detect conceptual rows that have been in either state for an abnormally long period of time and remove them. It is the responsibility of the DESCRIPTION clause of the status column to indicate what an abnormally long period of time would be.

The DISMAN-SCHEDULE-MIB's managed object schedAdminStatus demonstrates how to separate row control from row activation. Setting the schedAdminStatus to disabled(2) does not cause the row to be aged out/removed from the table.

Finally, a reasonable agent implementation must consider how many rows will be allowed to be created in the notReady/notInService state such that resources are not exhausted by an errant application.

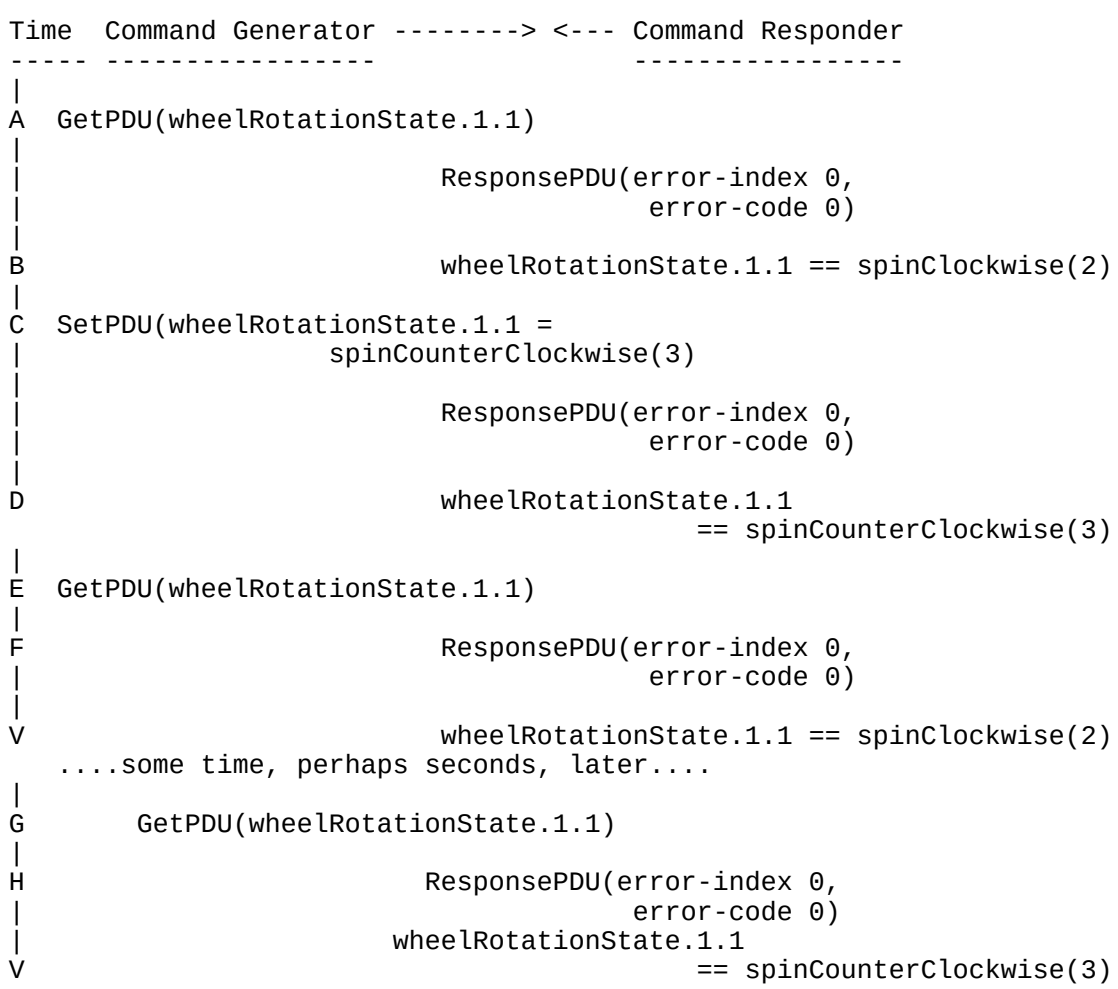
3.9. SET Operation Latency

Many standards track and enterprise MIB modules that contain read-write objects assume that an agent can complete a set operation as quickly as an agent can send back the status of the set operation to the application.

Consider the subtle operational shortcomings in the following object. It both reports the current state and allows a SET operation to change to a possibly new state.

```
wheelRotationState  OBJECT-TYPE
      SYNTAX        INTEGER { unknown(0),
                              idle(1),
                              spinClockwise(2),
                              spinCounterClockwise(3)
                            }
      MAX-ACCESS    read-write
      STATUS        current
      DESCRIPTION
      "The current state of a wheel."
  ::= { XXX 2 }
```

With the object defined, the following example represents one possible transaction.



The response to the GET request at time E will often confuse management applications that assume the state of the object should be spinCounterClockwise(3). In reality, the wheel is slowing down in order to come to the idle state then begin spinning counter clockwise.

This possibility of confusing and paradoxical interactions of administrative and operational state is inevitable when a single object type is used to control and report on both types of state. One common practice which we have already seen is to separate out the

desired (settable) state from current state. The objects ifAdminStatus and ifOperStatus from RFC 2863 [20] provide such an example of the separation of objects into desired and current state.

3.9.1. Subsystem Latency, Persistence Latency, and Activation Latency

A second way latency can be introduced in SET operations is caused by delay in agent implementations that must interact with loosely coupled subsystems. The time it takes the instrumented system to accept the new configuration information from the SNMP agent, process it and 'install' the updated configuration in the system or otherwise process the directives can often be longer than the SNMP response timeout.

In these cases, it is desirable to provide a "current state" object type which can be polled by the management application to determine the state of control of the loosely coupled subsystem which was affected by its configuration update.

More generally, some MIB objects may have high latencies associated with changes to their values. This could be either a function of saving the changed value to a persistent storage type, and/or activating a subsystem that inherently has high latency as discussed above. When defining such MIB objects, it might be wise to have the agent process set operations in the managed subsystem as soon as the Set PDU has been processed, and then update appropriate status objects when the save-to- persistent storage and (if applicable) activation has succeeded or is otherwise complete. Another approach would be to cause a notification to be sent that indicates that the operation has been completed.

When you describe an activation object, the DESCRIPTION clauses for these objects should give a hint about the likely latency for the completion of the operation. Keep in mind that from a management software perspective (as presented in the example of schedAdminStatus in Section 3.8.1), the combined latency of saving-to-persistence and activation are not distinguishable when they are part of a single operation.

3.10. Notifications and Error Reporting

For the purpose of this section, a 'notification' is as described in the SMIv2, RFC 2578 [2], by the NOTIFICATION-TYPE macro. Notifications can be sent in either SNMPv2c [19] or SNMPv3 TRAP or InformRequest PDUs. Given the sensitivity of configuration information, it is recommended that configuration operations always be performed using SNMPv3 due to its enhanced security capabilities. InformRequest PDUs should be used in preference to TRAP PDUs since

the recipient of the InformRequest PDUs responds with a Response PDU. This acknowledgment can be used to avoid unnecessary retransmission of NOTIFICATION-TYPE information when retransmissions are in fact required. The use of InformRequest PDUs (as opposed to TRAPs) is not at the control of the MIB module designer or agent implementor. The determination as to whether or not a TRAP or InformRequest PDU is sent from an SNMPv2c or SNMPv3 agent is generally a function of the agent's local configuration (but can be controlled with MIB objects in SNMPv3). To the extent notification timeout and retry values are determined by local configuration parameters, care should be taken to avoid unnecessary retransmission of InformRequest PDUs.

Configuration change and error information conveyed in InformRequest PDUs can be an important part of an effective SNMP-based management system. They also have the potential to be overused. This section offers some guidance for effective definition of NOTIFICATION-TYPE information about configuration changes that can be carried in InformRequest PDUs. Notifications can also play a key role for all kinds of error reporting from hardware failures to configuration and general policy errors. These types of notifications should be designed as described in Section 3.11 (Application Error Reporting).

3.10.1. Identifying Source of Configuration Changes

A NOTIFICATION-TYPE designed to report configuration changes should report the identity of the management entity initiating the configuration change. Specifically, if the entity is known to be a SNMP command generator, the transport address and SNMP parameters as found in table snmpTargetParamsTable from RFC 3413 SNMP-TARGET-MIB should be reported where possible. For reporting of configuration changes outside of the SNMP domain, the applicable change mechanism (for example, CLI vs. HTTP-based management client access) should be reported, along with whatever notion of "user ID" of the change initiator is applicable and available.

3.10.2. Limiting Unnecessary Transmission of Notifications

The design of event-driven synchronization models, essential to configuration management, can use notifications as an important enabling technique. Proper usage of notifications allows the manager's view of the managed element's configuration to be in close synchronization with the actual state of the configuration of the managed element.

When designing new NOTIFICATION-TYPEs, consider how to limit the number of notifications PDUs that will be sent with the notification information defined in the NOTIFICATION-TYPE in response to a configuration change or error event.

InformRequest PDUs, when compared to TRAP PDUs, have an inherent advantage when the concern is the reduction of unnecessary messages from the system generating the NOTIFICATION-TYPE data, when in fact retransmission of this data is required. That is, an InformRequest PDU is acknowledged by the receiving entity with a Response PDU. The receipt of this response allows the entity which generated the InformRequest PDU to verify (and record an audit entry, where such facilities exist on the agent system) that the message was received. As a matter of notification protocol, this receipt guarantee is not available when using TRAP PDUs, and if it is required, must be accomplished by the agent using some mechanism out of band to SNMP, and usually requiring the penalty of polling.

Regardless of the specific PDUs used to convey them, one way to limit the unnecessary generation of notifications is to include in the NOTIFICATION-TYPE definition situations where it need not be sent. A good example is the frDLCIStatusChange defined in FRAME-RELAY-DTE-MIB, RFC 2115 [21].

```
frDLCIStatusChange NOTIFICATION-TYPE
    OBJECTS  { frCircuitState }
    STATUS      current
    DESCRIPTION
       "This trap indicates that the indicated Virtual Circuit
       has changed state.  It has either been created or
       invalidated, or has toggled between the active and
       inactive states.  If, however, the reason for the state
       change is due to the DLCMI going down, per-DLCI traps
       should not be generated."
::= { frameRelayTraps 1 }
```

There are a number of other techniques which can be used to reduce the unwanted generation of NOTIFICATION-TYPE information. When defining notifications, the designer can specify a number of temporal limitations on the generation of specific instances of a NOTIFICATION-TYPE. For example, a definition could specify that messages will not be sent more frequently than once every 60 seconds while the condition which led to the generation of the notification persists. Alternately, a NOTIFICATION-TYPE DESCRIPTION clause could provide a fixed limit on the number of messages sent over the duration of the condition leading to sending the notification.

If NOTIFICATION-TYPE transmission is "aggregated" in some way - bounded either temporally or by absolute system state change as described above - the optimal design technique is to have the data delivered with the notification reference the actual number of underlying managed element transitions which brought about the notification. No matter which threshold is chosen to govern the

actual transmission of NOTIFICATION-TYPEs, the idea is to describe an aggregated event or related set of events in as few PDUs as possible.

3.10.3. Control of Notification Subsystem

There are standards track MIB modules that define objects that either augment or overlap control of notifications. For instance, FRAME-RELAY-DTE-MIB RFC 2115 defines frTrapMaxRate and DOCS-CABLE-DEVICE-MIB defines a set of objects in docsDevEvent that provide for rate limiting and filtering of notifications.

In the past, agents did not have a standard means to configure a notification generator. With the availability of the SNMP-NOTIFICATION-MIB module in RFC 3413 [9], it is strongly recommended that the filtering functions of this MIB module be used. This MIB facilitates the mapping of given NOTIFICATION-TYPEs and their intended recipients.

If the mechanisms of the SNMP-NOTIFICATION-MIB are not suitable for this application, a explanation of why they are not suitable should be included in the DESCRIPTION clause of any replacement control objects.

3.11. Application Error Reporting

MIB module designers should not rely on the SNMP protocol error reporting mechanisms alone to report application layer error state for objects that accept SET operations.

Most MIB modules that exist today provide very little detail as to why a configuration request has failed. Often the only information provided is via SNMP protocol errors which generally does not provide enough information about why an agent rejected a set request. Typically, there is an incumbent and sizable burden on the configuration application to determine if the configuration request failure is the result of a resource issue, a security issue, or an application error.

Ideally, when a "badValue" error occurs for a given set request, an application can query the agent for more details on the error. A badValue does not necessarily mean the command generator sent bad data. An agent could be at fault. Additional detailed diagnostic information may aid in diagnosing conditions in the integrated system.

Consider the requirement of conveying error information about a MIB expression 'object' set within the DISMAN-EXPRESSION-MIB [40] that occurs when the expression is evaluated. Clearly, none of the available protocol errors are relevant when reporting an error condition that occurs when an expression is evaluated. Instead, the DISMAN-EXPRESSION-MIB provides objects to report such errors (the expErrorTable). Instead, the expErrorTable maintains information about errors that occur at evaluation time:

```
 expErrorEntry OBJECT-TYPE
     SYNTAX      ExpErrorEntry
     MAX-ACCESS  not-accessible
     STATUS      current
     DESCRIPTION
     "Information about errors in processing an expression.
     Entries appear in this table only when there is a matching
     expExpressionEntry and then only when there has been an
     error for that expression as reflected by the error codes
     defined for expErrorCode."
INDEX       { expExpressionOwner, expExpressionName }
```

More specifically, a MIB module can provide configuration applications with information about errors on the managed device by creating columnar object types in log tables that contain error information particular to errors that occur on row activation.

Notifications with detailed failure information objects can also be used to signal configuration failures. If this approach is used, the configuration of destinations for NOTIFICATION-TYPE data generated from configuration failures should be considered independently of the those for other NOTIFICATION-TYPEs which are generated for other operational reasons. In other words, in many management environments, the network operators interested in NOTIFICATION-TYPEs generated from configuration failures may not completely overlap with the community of network operators interested in NOTIFICATION-TYPEs generated from, for example, network interface failures.

3.12. Designing MIB Modules for Multiple Managers

When designing a MIB module for configuration, there are several pertinent considerations to provide support for multiple managers.

The first is to avoid any race conditions between two or more authorized management applications issuing SET protocol operations spanning over more than a single PDU.

The standard textual convention document [3] defines TestAndIncr, often called a spinlock, which is used to avoid race conditions.

A MIB module designer may explicitly define a synchronization object of syntax TestAndIncr or may choose to rely on snmpSetSerialNo (a global spinlock object) as defined in SNMPv2-MIB.

```
snmpSetSerialNo OBJECT-TYPE
SYNTAX     TestAndIncr
MAX-ACCESS read-write
STATUS     current
DESCRIPTION
        "An advisory lock used to allow several cooperating
        command generator applications to coordinate their
        use of the SNMP set operation.

        This object is used for coarse-grain coordination.
        To achieve fine-grain coordination, one or more similar
        objects might be defined within each MIB group, as
        appropriate."
::= { snmpSet 1 }
```

Another prominent TestAndIncr example can be found in the SNMP-TARGET- MIB [9], snmpTargetSpinLock.

Secondly, an agent should be able to report configuration as set by different entities as distinguishable from configuration defined external to the SNMP domain, such as application of a default or through an alternate management interface like a command line interface. Section 3.10.1 describes considerations for this practice when designing NOTIFICATION-TYPEs. The OwnerString textual convention from RMON-MIB RFC 2819 [32] has been used successfully for this purpose. More recently, RFC 3411 [1] introduced the SnmpAdminString which has been designed as a UTF8 string. This is more suitable for representing names in many languages.

Experience has shown that usage of OwnerString to represent row ownership can be a useful diagnostic tool as well. Specifically, the use of the string "monitor" to identify configuration set by an agent/local management has been prevalent and useful in applications.

Thirdly, consider whether there is a need for multiple managers to configure the same set of tables. If so, an "OwnerString" may be used as the first component of a table's index to allow VACM to be used to protect access to subsets of rows, at least at the level of securityName or groupName provided. RFC 3231 [25], Section 6 presents this technique in detail. This technique does add complexity to the managed device and to the configuration management application since the manager will need to be aware of these additional columnar objects in configuration tables and act appropriately to set them. Additionally, the agent must be

configured to provide the appropriate instance-level restrictions on the modifiability of the instances.

3.13. Other MIB Module Design Issues

3.13.1. Octet String Aggregations

The OCTET STRING syntax can be used as an extremely flexible and useful datatype when defining managed objects that allow SET operation. An octet string is capable of modeling many things and is limited in size to 65535 octets by SMIv2[2].

Since OCTET STRINGS are very flexible, the need to make them useful to applications requires careful definition. Otherwise, applications will at most simply be able to display and set them.

Consider the following object from RFC 3418 SNMPv2-MIB [11].

```
sysLocation OBJECT-TYPE
SYNTAX      DisplayString (SIZE (0..255))
MAX-ACCESS  read-write
STATUS      current
DESCRIPTION
        "The physical location of this node (e.g., 'telephone
        closet, 3rd floor'). If the location is unknown, the value
        is the zero-length string."
::= { system 6 }
```

Such informational object types have come to be colloquially known as "scratch pad objects". While often useful, should an application be required to do more with this information than be able to read and set the value of this object, a more precise definition of the contents of the OCTET STRING is needed, since the actual format of an instance for such an object is unstructured. Hence, alternatively, dividing the object type into several object type definitions can provide the required additional structural detail.

When using OCTET STRINGS, avoid platform dependent data formats. Also avoid using OCTET STRINGS where a more precise SMI syntax such as SnmpAdminString or BITS would work.

There are many MIB modules that attempt to optimize the amount of data sent/received in a SET/GET PDU by packing octet strings with aggregate data. For example, the PortList syntax as defined in the Q-BRIDGE-MIB (RFC 2674 [26]) is defined as follows:

```
PortList ::= TEXTUAL-CONVENTION
 STATUS      current
 DESCRIPTION
     "Each octet within this value specifies a set of eight
     ports, with the first octet specifying ports 1 through
     8, the second octet specifying ports 9 through 16, etc.
     Within each octet, the most significant bit represents
     the lowest numbered port, and the least significant bit
     represents the highest numbered port.  Thus, each port
     of the bridge is represented by a single bit within the
     value of this object.  If that bit has a value of '1'
     then that port is included in the set of ports; the port
     is not included if its bit has a value of '0'."
 SYNTAX      OCTET STRING
```

This compact representation saves on data transfer but has some limitations. Such complex instance information is difficult to reference outside of the object or use as an index to a table. Additionally, with this approach, if a value within the aggregate requires change, the entire aggregated object instance must be written.

Providing an SNMP table to represent aggregate data avoids the limitations of encoding data into OCTET STRINGs and is thus the better general practice.

Finally, as previously mentioned in Section 3.3.6.3, one should consider the practical ramifications of instance transfer for object types of SYNTAX OCTET STRING where they have typical instance data requirements close to the upper boundary of SMIv2 OCTET STRING instance encoding. Where such object types are truly necessary at all, SNMP/UDP may not be a very scalable means of transfer and alternatives should be explored.

3.13.2. Supporting multiple instances of a MIB Module

When defining new MIB modules, one should consider if there could ever be multiple instances of this MIB module in a single SNMP entity.

MIB modules exist that assume a one to many relationship, such as MIBs for routing protocols which can accommodate multiple "processes" of the underlying protocol and its administrative framework. However, the majority of MIB modules assume a one-to-one relationship between the objects found in the MIB module and how many instances will exist on a given SNMP agent. The OSPF-MIB, IP-MIB, BRIDGE-MIB are all examples that are defined for a single instance of the technology.

It is clear that single instancing of these MIB modules limits implementations that might support multiple instances of OSPF, IP stacks or logical bridges.

In such cases, the ENTITY-MIB [RFC2737] can provide a means for supporting the one-to-many relationship through naming scopes using the entLogicalTable. Keep in mind, however, that there are some drawbacks to this approach.

1) One cannot issue a PDU request that spans naming scopes. For example, given two instances of BRIDGE-MIB active in a single agent, one PDU cannot contain a request for dot1dBaseNumPorts from both the first and second instances.

2) Reliance on this technique creates a dependency on the Entity MIB for an application to be able to access multiple instances of information.

Alternately, completely independently of the Entity MIB, multiple MIB module instances can be scoped by different SNMP contexts. This does, however, require the coordination of this technique with the administrative establishment of contexts in the configured agent system.

3.13.3. Use of Special Optional Clauses

When defining integer-based objects for read-create, read-write and read-only semantics, using the UNITS clause is recommended in addition to specification in the DESCRIPTION clause of any particular details of how UNITs are to be interpreted.

The REFERENCE clause is also recommended as a way to help an implementer track down related information on a given object. By adding a REFERENCE clause to the specific underlying technology document, multiple separate implementations will be more likely to interoperate.

4. Implementing SNMP Configuration Agents

4.1. Operational Consistency

Successful deployment of SNMP configuration systems depends on understanding the roles of MIB module design and agent design.

Both module and agent design need to be undertaken with an understanding of how UDP/IP-based SNMP behaves. A current practice in MIB design is to consider the idempotency of settable objects. Idempotency basically means being able to invoke the same set operation repeatedly but resulting in only a single activation.

Here is an example of the idempotency in action:

```
Manager                              Agent
--------                             ------
Set1 (Object A, Value B) --->        receives set OK and responds
                          X<-------- Response PDU(OK) is dropped by
                                     network
Manager times out
and sends again
Set2 (Object A, Value B) --->       receives set OK (does nothing),
                                    responds
                          <-------- with a Response PDU(OK)
Manager receives OK
```

Had object A been defined in a stateful way, the set operation might have caused the Set2 operation to fail as a result of interaction with Set1. If the agent implementation is not aware of such a possible situation on the second request, the agent may behave poorly by performing the set request again rather than doing nothing.

The example above shows that all of the software that runs on a managed element and in managed applications should be designed in concert when possible. Particular emphasis should be placed at the logical boundaries of the management system components in order to ensure correct operation.

1. The first interface is between SNMP agents in managed devices and the management applications themselves. The MIB document is a contract between these two entities that defines expected behavior - it is a type of API.

2. The second interface is between the agent and the instrumented subsystem. In some cases, the instrumented subsystem will require modification to allow for the dynamic nature of SNMP-based configuration, control and monitoring operations. Agent implementors must also be sensitive to the operational code and device in order to minimize the impact of management on the primary subsystems.

Additionally, while the SNMP protocol-level and MIB module-level modeling of configuration operations may be idempotent and stateless from one set operation to another, it may not be that way in the

underlying subsystem. It is possible that an agent may need to manage this state in these subsystem architectures explicitly when it has placed the underlying subsystem into an "intermediate" state at a point in processing a series of SET PDUs. Alternatively, depending on the underlying subsystem in question, the agent may be able to buffer all of the configuration set operations prior to activating them in the subsystem all at once (to accommodate the nature of the subsystem).

As an example, it would be reasonable to define a MIB module to control Virtual Private Network (VPN) forwarding, in which a management station could set a set of ingress/egress IP addresses for the VPN gateway. Perhaps the MIB module presumes that the level of transactionality is the establishment of a single row in a table defining the address of the ingress/egress gateway, along with some prefix information to assist in routing at the VPN layer to that gateway. However, it would be conceivable that in an underlying Layer 2 VPN subsystem instrumentation, the requirement is that all existing gateways for a VPN be deleted before a new one can be defined--that, in other words, in order to add a new gateway, g(n), to a VPN, gateways g(1)..g(n-1) need to be removed, and then all n gateways reestablished with the VPN forwarding service. In this case, one could imagine an agent which has some sort of timer to establish a bounded window for receipt of SETs for new VPN gateways, and to activate them in this removal-then-reestablishment of existing and new gateways at the end of this window.

4.2. Handling Multiple Managers

Devices are often modified by multiple management entities and with different management techniques. It is sometimes the case that an element is managed by different organizations such as when a device sits between administrative domains.

There are a variety of approaches that management software can use to ensure synchronization of information between the manager(s) and the managed elements.

An agent should report configuration changes performed by different entities. It should also distinguish configuration defined locally such as a default or locally specified configuration made through an alternate management interface such as a command line interface. When a change has been made to the system via SNMP, CLI, or other method, a managed element should send an notification to the manager(s) configured as recipients of these applicable notifications. These management applications should update their

local configuration repositories and then take whatever additional action is appropriate. This approach can also be an early warning of undesired configuration changes.

Managers should also develop mechanisms to ensure that they are synchronized with each other.

4.3. Specifying Row Modifiability

Once a RowStatus value is active(1) for a given row, the management application should be able to determine what the semantics are for making additional changes to a row. The RMON MIB control table objects spell out explicitly what managed objects in a row can and cannot be changed once a given RowStatus goes active.

As described earlier, some operations take some time to complete. Some systems also require that they remain in a particular state for some period before moving to another. In some cases, a change to one value may require re-initialization of the system. In all of these cases, the DESCRIPTION clause should contain information about requirements of the managed system and special restrictions that managers should observe.

4.4. Implementing Write-only Access Objects

The second version of the SNMP SMI dropped direct support for a write-only object. It is therefore necessary to return something when reading an object that you may have wished to have write-only semantics. Such objects should have a DESCRIPTION clause that details what the return values should be. However, regardless of the approach, the value returned when reading the object instance should be meaningful in the context of the object's semantics.

5. Designing Configuration Management Software

In this section, we describe practices that should be used when creating and deploying management software that configures one or more systems using SNMP. Functions all configuration management software should provide, regardless of the method used to convey configuration information to the managed systems are backup, fail-over, and restoration. A management system should have the following features:

1. A method for restoring a previous configuration to one or more devices. Ideally this restoration should be time indexed so that a network can be restored to a configured state as of a specific time and date.

2. A method for saving back up versions of the configuration data in case of hardware or software failure.

3. A method of providing fail-over to a secondary (management) system in case of a primary failure. This capability should be deployed in such a way that it does not cause duplicate polling of configuration.

These three capabilities are of course important for other types of management that are not the focus of this BCP.

5.1. Configuration Application Interactions with Managed Systems

From the point of view of the design of the management application, there are three basic requirements to evaluate relevant to SNMP protocol operations and configuration:

- o Set and configuration activation operations
- o Notifications from the device
- o Data retrieval and collection

Depending on the requirements of the specific services being configured, many other requirements may, and probably will, also be present.

The design of the system should not assume that the objects in a device that represent configuration data will remain unchanged over time.

As standard MIB modules evolve and vendors add private extensions, the specific configuration parameters for a given operation are likely to change over time. Even in the case of a configuration application that is designed for a single vendor, the management application should allow for variability in the MIB objects that will be used to configure the device for a particular purpose. The best method to accomplish this is by separating, as much as possible, the operational semantics of a configuration operation from the actual data. One way that some applications achieve this is by having the specific configuration objects that are associated with a particular device be table driven rather than hard coded. Ideally, management software should verify the support in the devices it is expected to manage and report any unexpected deviations to the operator. This approach is particularly valuable when developing applications that are intended to support equipment or software from multiple vendors.

5.1.1. SET Operations

Management software should be mindful of the environment in which SET operations are being deployed. The intent here is to move configuration information as efficiently as possible to the managed device. There are many ways to achieve efficiency and some are specific to given devices. One general case that all management software should employ is to reduce the number of SET PDU exchanges between the managed device and the management software to the smallest reasonable number. One approach to this is to verify the largest number of variable bindings that can fit into a SET PDU for a managed device. In some cases, the number of variable bindings to be sent in a particular PDU will be influenced by the device, the specific MIB objects and other factors.

Maximizing the number of variable bindings in a SET PDU also has benefits in the area of management application transaction initiation, as we will discuss in the following section.

There are, though, agents that may have implementation limitations on the number and order of varbinds they can handle in a single SET PDU. In this case, sending fewer varbinds will be necessary.

As stated at the outset of this section, the management application software designer must be sensitive to the design of the SNMP software in the managed device. For example, the software in the managed device may require that all that all related configuration information for an operation be conveyed in a single PDU because it has no concept of a transaction beyond a single SNMP PDU. Another example has to do with the RowStatus textual convention. Some SNMP agents implement a subset of the features available and as such the management application must avoid using features that may not be supported in a specific table implementation (such as createAndWait).

5.1.2. Configuration Transactions

There are several types of configuration transactions that can be supported by SNMP-based configuration applications. They include transactions on a scalar object, transactions in a single table (within and across row instances), transactions across several tables in a managed device and transactions across many devices. The manager’s ability to support these different transactions is partly dependent on the design of the MIB objects used in the configuration operation.

To make use of any kind of transaction semantics effectively, SNMP management software must be aware of the information in the MIB modules that it is to configure so that it can effectively utilize

RowStatus objects for the control of transactions on one or more tables. Such software must also be aware of control tables that the device supports that are used to control the status of one or more other tables.

To the greatest extent possible, the management application should provide the facility to support transactions across multiple devices. This means that if a configuration operation is desired across multiple devices, the manager can coordinate these configuration operations such that they become active as close to simultaneously as possible.

Several practical means are present in the SNMP model that support management application level transactions. One was mentioned in the preceding section, that transactions can be optimized by including the maximum number of SET variable bindings possible in a single PDU sent to the agent.

There is an important refinement to this. The set of read-create row data objects for tables should be sent in a single PDU, and only placed across multiple PDUs if absolutely necessary. The success of these set operations should be verified through the response(s) to the Set PDU or subsequent polling of the row data objects. The applicable RowStatus object(s), may be set to active only after this verification. This is the only tractable means of affording an opportunity for per-row rollback, particularly when the configuration change is across table row instances on multiple managed devices.

Finally, where a MIB module exposes the kind of helpful transaction management object types that were discussed in Section 3.3.5, it is clearly beneficial to the integrity of the management application's capacity to handle transactions to make use of them.

5.1.3. Tracking Configuration Changes

As previously described in Section 3.3.5 (Summary Objects and State Tracking), agents should provide the capability for notifications to be sent to their configured management systems whenever a configuration operation is completed or is detected to have failed. The management application must be prepared to accept these notifications so that it knows the current configured state of the devices under its control. Upon receipt of the notification, the management application should use getBulk or getNext to retrieve the configuration from the agent and store the relevant contents in the management application database. The GetBulkRequest-PDU is useful for this whenever supported by the managed device, since it is more efficient than the GetNextRequest-PDU when retrieving large amounts

of data. For the purposes of backward compatibility, the management station should also support and make use of the GetNextRequest-PDU when the agent does not support the GetBulkRequest-PDU.

Management systems should also provide configuration options with defaults for users that tend to retrieve the smallest amount of data to achieve the particular goal of the application, to avoid unnecessary load on managed devices for the most common retrieval operations.

5.1.4. Scalability of Data Retrieval

The techniques for efficient data retrieval described in the preceding sections comprise only one aspect of what application developers should consider in this regard when developing configuration applications. Management applications should provide for distributed processing of the configuration operations. This also extends to management functions that are not the focus of this document. Techniques of distributed processing can also be used to provide resilience in the case of network failures. An SNMP-based configuration management system might be deployed in a distributed fashion where three systems in different locations keep each other synchronized. This synchronization can be accomplished without additional polling of network devices through a variety of techniques. In the case of a failure, a 'backup' system can take over the configuration responsibilities from the failed manager without having to re-synchronize with the managed elements since it will already be up to date.

6. Deployment and Security Issues

Now that we have considered the design of SNMP MIB data for configuration, agent implementation of its access, and management application issues in configuration using SNMP, we turn to a variety of operational considerations which transcend all three areas.

6.1. Basic assumptions about Configuration

The following basic assumptions are made about real world configuration models.

1) Operations must understand and must be trained in the operation of a given technology. No configuration system can prevent an untrained operator from causing outages due to misconfiguration.

2) Systems undergoing configuration changes must be able to cope with unexpected loss of communication at any time.

During configuration operations, network elements must take appropriate measures to leave the configuration in a consistent/recognizable state by either rolling back to a previously valid state or changing to a well-defined or default state.

3) Configuration exists on a scale from relatively unchanging to a high volume, high rate of change. The former is often referred to as "set and forget" to indicate that the configuration changes quite infrequently. The latter, "near real-time change control" implies a high frequency of configuration change. Design of configuration management must take into account the rate and volume of change expected in a given configuration subsystem.

6.2. Secure Agent Considerations

Vendors should not ship a device with a community string 'public' or 'private', and agents should not define default community strings except when needed to bootstrap devices that do not have secondary management interfaces. Defaults lead to security issues that have been recognized and exploited. When using SNMPv1, supporting read-only community strings is a common practice.

Version 3 of the SNMP represents the current standard for the Internet Management Framework and is recommended for all network management applications. In particular, SNMPv3 provides authorization, authentication, and confidentiality protection and is essential to meeting the security considerations for all management of devices that support SNMP-based configuration.

6.3. Authentication Notifications

The default state of RFC 1215 [17] Authentication notifications should be off. One does not want to risk accidentally sending out authentication failure information, which by itself could constitute a security liability. Enabling authentication Notifications should be done in the context of a management security scheme which considers the proper recipients of this information.

There are other liabilities where authentication notifications are generated without proper security infrastructure. When notifications are sent in SNMPv1 trap PDUs, unsolicited packets to a device can causes one or more trap PDUs to be created and sent to management stations. If these traps flow on shared access media and links, the community string from the trap may be gleaned and exploited to gain access to the device. At the very least, this risk should be mitigated by having the authentication trap PDU be conveyed with a

community string which is only used for authentication traps from the agent, and would be useless for access inbound to the agent to get at other management data.

A further liability of authentication traps can be seen when they are being generated in the face of a Denial Of Service (DOS) attack, in the form of a flood of PDUs with invalid community strings, on the agent system. If it is bad enough that the system is having to respond to and recover from the invalid agent data accesses, but the problem will be compounded if a separate Authentication notification PDU is sent to each recipient on the management network.

6.4. Sensitive Information Handling

Some MIB modules contain objects that may contain data for keys, passwords and other such sensitive information and hence must be protected from unauthorized access. MIB documents that are created in the IETF must have a 'Security Considerations' section, which details how sensitive information should be protected. Similarly, MIB module designers who create MIB documents for private MIB objects should include similar information so that users of the products containing these objects can take appropriate precautions.

Even if a device does support DES, it should be noted that configuration of keys for other protocols via SNMP Sets protected by DES should not be allowed if the other keys are longer than the 56 bit DES keys protecting the SNMP transmission.

The DESCRIPTION clause for these object types and their Security Considerations sections in the documents which define them should make it clear how and why these specific objects are sensitive and that a user should only make them accessible for encrypted SNMP access. Vendors should also document sensitive objects in a similar fashion.

Confidentiality is not a mandatory portion of the SNMPv3 management framework [6].

Prior to SNMPv3, providing customized views of MIB module data was difficult. This led to objects being defined such as the following from [41].

```
docsDevNmAccessEntry OBJECT-TYPE
    SYNTAX      DocsDevNmAccessEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "An entry describing access to SNMP objects by a
         particular network management station.  An entry in
         this table is not readable unless the management station
         has read-write permission (either implicit if the table
         is empty, or explicit through an entry in this table.
         Entries are ordered by docsDevNmAccessIndex.  The first
         matching entry (e.g., matching IP address and community
         string) is used to derive access."
    INDEX { docsDevNmAccessIndex  }
    ::= {  docsDevNmAccessTable 1 }
```

New MIB modules should capitalize on existing security capabilities of SNMPv3 Framework. One way they can do this is by indicating the level of security appropriate to different object types. For example, objects that change the configuration of the system might be protected by using the authentication mechanisms in SNMPv3. Specifically, it is useful to design MIB module object grouping with considerations for VACM views definition, such that users can define and properly scope what tables are visible to a given user and view.

7. Policy-based Management

In some designs and implementations, a common practice used to move large amounts of data involves using SNMP as a control channel in combination with other protocols defined for transporting bulk data. This approach is sub-optimal since it raises a number of security and other concerns. Transferring large amounts of configuration data via SNMP can be efficiently performed with several of the techniques described earlier in this document. This policy section shows how even greater efficiency can be achieved using a set of relatively new design mechanisms. This section gives background and defines terms that are relevant to this field and describes some deployment approaches.

7.1. What Is the Meaning of 'Policy-based'?

In the past few years of output from standards organizations and networking vendor marketing departments, the term 'policy' has been heavily used, touted, and contorted in meaning. The result is that the true meaning of 'policy' is unclear without greater qualification where it is used.

[42] gives the term 'policy' two explicit definitions:

- A definite goal, course or method of action to guide and determine present and future decisions. "Policies" are implemented or executed within a particular context (such as policies defined within a business unit).

- Policies as a set of rules to administer, manage, and control access to network resources.

  Note that these two views are not contradictory since individual rules may be defined in support of business goals.

As it pertains to our discussion of the term 'policy-based configuration', the meaning is significantly more specific. In this context, we refer to a way of integrating data and the management actions which use it in such a way that:

- there is the ability to specify "default" configuration data for a number of instances of managed elements, where those instances can be correlated in some data driven or algorithmic way. The engine to do this correlation and activate instances from defaults may reside in the agent or externally. Where the representation of these defaults are in the MIB design itself, the object types supporting this notion are referred to as "template objects".

- the activation of instance data derived from template object types results from minimal activation directives from the management application, once the instances of the template object types have been established.

- somewhat independently, the architecture of the overall management agent may accommodate the definition and evaluation of management and configuration policies. The side-effects of the evaluation of these policies typically include the activation of certain configuration directives. Where management data design exposes template object types, the policy-driven activation can (and ideally, should) include the application of template object instances to the analogous managed element instance-level values.

As it pertains to template object data, the underlying notions implied here have been prevalent for some time in non-SNMP management regimes. A common feature of many command line interfaces for configuring routers is the specification of one or more access control lists. These typically provide a set of IP prefixes, BGP autonomous system numbers, or other such identifying constructs (see, for example, [42]). Once these access control lists are assembled, their application to various interfaces, routing processes, and the like are specified typically in the configuration of what the access control list is applied to. Consistent with the prior properties to

define our use of policy-based configuration, a) the access list is defined independent from its point of application, and b) its application is independent of the access list definition. For example, changing the application of an access list from one interface to the other does not require a change in the access list itself. The first point just mentioned suggests what is necessary for template-based data organization. The second suggests its application in a policy-based manner.

Let us now examine the motivation for such a system or subsystem (perhaps bounded at the level of a 'template-enabled' MIB module, given the above definition). Let us explore the importance of policy-based techniques to configuration specifically.

7.2. Organization of Data in an SNMP-Based Policy System

The number of configurable parameters and 'instances' such as interfaces has increased as equipment has become larger and more complex.

At the same time, there is a need to configure many of these systems to operate in a coordinated fashion. This enables the delivery of new specialized services that require this coordinated configuration. Examples include delivery of virtual private networks and connections that guarantee specific service levels.

The growth in size and complexity of configuration information has significant implications for its organization as well as its efficient transfer to the management agent. As an example, an agent that implements the Bridge MIB [24] could be used to represent a large VLAN with some 65,000 port entries. Configuring such a VLAN would require the establishment of dot1dStpPortTable and dot1DStaticTable entries for each such virtual port. Each table entry would contain several parameters. A more efficient approach is to provide default values for the creation of new entries that are appropriate to the VLAN environment in our example. The local management infrastructure should then iterate across the system setting the default values to the selected ports as groups.

To date, this kind of large-scale configuration has been accomplished with file transfer, by setting individual MIB objects, or with many CLI commands. In each of these approaches the details for each instance are contained in the file, CLI commands or MIB objects. That is, they contain not only the value, and type of object, but also the exact instance of the object to which to apply the value. It is this property that tends to make configuration operations explode as the number of instances (such as interfaces) grows. This per-instance approach can work for a few machines configured by

experts, but there is a need for a more scalable solution. Template-based data organization and policy-based management abstracts the details above the instance level, which means that fewer SET requests are sent to a managed device.

Realization of such a policy-driven system requires agents that can take defaults and apply them to instances based on a rule that defines under what conditions the defaults (policy) are to be applied. A policy-driven configuration system which is to be scalable needs to expose a means of layering its application of defaults at discrete ranges of granularity. The spectrum of that granularity might have a starting hierarchy point to apply defaults at the breadth of a network service.

Ultimately, such a layering ends up with features to support instance-level object instance data within the running agent.

An example of this kind of layering is implicit in the principle of operations of a SNMPCONF Policy-Based Management MIB [36] (PM-MIB) implementation. However, other entity management systems have been employing these kinds of techniques end-to-end for some time, in some cases using SNMP, in some cases using other encodings and transfer technologies. What the PM-MIB seeks to establish, in an environment ideal for its deployment, is an adaptation between MIB module data which was not designed using template object types, and the ability to allow the PM-MIB agent engine to apply instances of that data as though it were template-based.

7.3. Information Related to Policy-based Configuration

In order for effective policy management to take place, a range of information about the network elements is needed to avoid making poor policy decisions. Even in those cases where policy-based configuration is not in use, much of the information described in this section can be useful input to the decision-making process about what type of configuration operations to do.

For this discussion it is important to make distinctions between distribution of policy to a system, activation of a policy in a system, and changes/failures that take place during the time the policy is expected to be active. For example, if an interface is down that is included in a policy that is distributed, there may not be an error since the policy may not be scheduled for activation until a later time.

On the other hand, if a policy is distributed and applied to an interface that should be operational and it is not, clearly this is a problem, although it is not an error in the configuration policy

itself. With this as background, here are some areas to consider that are important to making good policy configuration decisions and establishing when a policy has 'failed'.

- o The operational state of network elements that are to be configured.

  Care should be taken to determine if the sub-components to be configured are available for use. In some cases the elements may not be available. The policy configuration software should determine if this is a prerequisite to policy installation or if the condition is even acceptable. This decision is separate from the one to be made about policy activation. Installation is when the policy is sent from the policy manager to the managed device and activation is turning on the policy. In those cases where policy is distributed when the sub-component such as an interface or disk is not available, the managed system should send a notification to the designated management station when the policy is to become active or if the resource is still not available.

- o The capabilities of the devices in the network.

  A capability can be almost any unit of work a network element can perform. These include routing protocols supported, Web server and OS versions, queuing mechanisms supported on each interface that can be used to support different qualities of service, and many others. This information can be obtained from the capabilities table of the Policy MIB module [36].

  Historically, management applications have had to obtain this type of information by issuing get requests for objects they might want to use. This approach is far less efficient since it requires many get requests and is more error prone since some instances will not exist until configured. The new capabilities table is an improvement on the current technique.

- o The capacity of the devices to perform the desired work.

  Capability is an ability to perform the desired work while a capacity is a measure of how much of that capability the system has. The policy configuration application should, wherever possible, evaluate the capacity of the network element to perform the work identified by the policy. In some systems it will not be possible to obtain the capacity of the managed elements to perform the desired work directly, even though it may be possible to monitor the amount of work the element performs. In these cases, the management application may benefit from pre-configured

information about the capacity of different network elements so that evaluations of the resources available can be done before distributing new policies.

Utilization refers to how much capacity for a particular capability has been consumed. For devices that have been under policy configuration control for any period of time, a certain percentage of the available capacity of the managed elements will be used. Policies should not be distributed to systems that do not have the resources to carry out the policy in a reasonable period of time.

7.4. Schedule and Time Issues

This section applies equally to systems that are not policy-based as well as policy-based systems, since configuration operations often need to be synchronized across time zones. Wherever possible, the network elements should support time information using the standard DateAndTime TC that includes local time zone information. Policy-based management often requires more complex time expressions than can be conveyed with the DateAndTime TC. See the Policy-Based Management MIB document [36] for more information. Some deployed systems do not store complex notions of local time and thus may not be able to process policy directives properly that contain time zone relevant data. For this reason, policy management applications should have the ability to ascertain the time keeping abilities of the managed system and make adjustments to the policy for those systems that are time-zone challenged.

7.5. Conflict Detection, Resolution and Error Reporting

Policies sent to a device may contain conflicting instructions. Detection of such commands can occur at the device or management level and may be resolved using any number of mechanisms (examples are, last configuration set wins, or abort change). These unintended conflicts should be reported. Conflicts can occur at different levels in a chain of commands. Each 'layer' in policy management system should be able to check for some errors and report them. This is conceptually identical to programs raising an exception and passing that information on to software that can do something meaningful with it.

At the instance level, conflict detection has been performed in a limited way for some time in software that realizes MIB objects at this level of resolution. This detection is independent of policy. The types of 'conflicts' usually checked for are resource availability and validity of the set operations. In a policy enabled

system, there are no additional requirements for this software assuming that good error detection and reporting appropriate to this level have already been implemented.

7.5.1. Changes to Configuration Outside of the Policy System

It is essential to consider changes to configuration that are initiated outside of the policy system. A goal of SNMP-based policy management is to coexist with other kinds of management software that have historically been instance based management. The best example is the command line interface.

A notification should be sent whenever an out-of-policy control change is made to an element that is under the control of policy. This notification should include the policy that was affected, the instance of the element that was changed and the object and value that it was changed to.

Even for those systems that have no concept of policy control, the ideas presented above make sense. That is, if SNMP co-exists with other access methods such as a CLI, it is essential that the management station remain synchronized with changes that might have been made to the managed device using other methods. As a result, the approach of sending a notification when another access method makes a change is a good one. Of course this should be configurable by the user.

7.6. More about Notifications in a Policy System

Notifications can be useful in determining a failure of a policy as a result of an error in the policy or element(s) under policy control. As with all notifications, they should be defined and controlled in such a way that they do not create a problem by sending more than are helpful over a specific period of time. For example, if a policy is controlling 1,000 interfaces and fails, one notification rather than 1,000 may be the better approach. In addition, such notifications should be defined to include as much information as possible to aid in problem resolution.

7.7. Using Policy to Move Less Configuration Data

One of the advantages of policy-based configuration with SNMP is that many configuration operations can be conveyed with a small amount of data. Changing a single configuration parameter for each of 100 interfaces on a system might require 100 CLI commands or 100 SNMP variable bindings using conventional techniques.

Using policy-based configuration with SNMP, a single SET PDU can be sent with the policy information necessary to apply a configuration change to 100 similar interfaces. This efficiency gain is the result of eliminating the need to send the value for each instance to be configured. The 'default' for each of the instances included in the policy is sent, and the rule for selection of the instances that the default is to be applied to can also be carried (see the Policy MIB module [36]).

To extend the example above, assume that there are 10 parameters that need to change. Using conventional techniques, there would now be 1,000 variable bindings, one for each instance of each new value for each interface. Using policy-based configuration with SNMP, it is still likely that all the information can be conveyed in one SET PDU. The only difference in this case is that there are ten parameters sent that will be the 'template' used to create instances on the managed interfaces.

This efficiency gain not only applies to SET operations, but also to those management operations that require configuration information. Since the policy is also held in the storage for cross-instance defaults (for example, the pmPolicyTable in [36]), an entire data set that potentially controls hundreds of rows of information can be retrieved in a single GET request.

A policy-friendly data organization such as this is consistent and integrates well with MIB module objects which support "summary" activation and activation reporting, of the kind discussed in Section 3.3.5.

8. Example MIB Module With Template-based Data

This section defines a MIB module that controls the heating and air conditioning system for a large building. It contains both configuration and counter objects that allow operators to see how much cooling or heating a particular configuration has consumed. Objects that represent the configuration information at a "default" level (as referenced above) are also included.

These tables, in combination with the application of the tables' row instance data as templated 'defaults', will allow operators to configure and monitor many rooms at once, change the configuration parameters based on time of day, and make a number of other sophisticated decisions based on the 'policy' implied by these defaults and their application. For this reason, these configuration controls have their instances specified from template object types.

In our simplified Heating Ventilation and Air Conditioning (HVAC) model we will create three tables based on a simple analysis. More complicated systems will need more tables, but the principles will be the same.

Step 1: As with any other MIB module design, the first step is to determine what objects are necessary for configuration and control operations. The first table to be created is a fairly traditional monitoring table. It includes indices so that we will know what rooms the counters and status objects are for. It includes an object that is a RowPointer to a table that contains configuration information. The objects for the bldgHVACTable, our first table in the HVAC MIB module are:

Index objects that identify what floor and office we are managing:

bldgHVACFloor
bldgHVACOffice

A single index reference to a table that 'glues' configuration information defaults with descriptive information:

bldgHVACCfgTemplate

A set of objects that show status and units of work (bldgHVACCoolOrHeatMins) and standard per-row SnmpAdminString, StorageType, and RowStatus columnar objects:

bldgHVACFanSpeed
bldgHVACCurrentTemp
bldgHVACCoolOrHeatMins
bldgHVACDiscontinuityTime
bldgHVACOwner
bldgHVACStatus

Step 2: A configuration description table. The purpose of this table is to provide a unique string identifier for templates. These may be driven by policies in a network. If it were necessary to configure devices to deliver a particular quality of service, the index string of this table could be the name and the description part, it could be a brief description of the underlying motivation such as: "provides extra heat to corner offices to counteract excessive exterior wind

chill". Standard owner and status objects may also be helpful and are included here. The row columnar objects are:

bldgHVACCfgTemplateInfoIndex
bldgHVACCfgTemplateInfoID
bldgHVACCfgTemplateInfoDescr
bldgHVACCfgTemplateInfoOwner
bldgHVACCfgTemplateInfoStatus

Notice that to this point we have provided no configuration information. That will be in the next table. Some readers may wonder why this table is not combined with the configuration template table described in the next step. In fact, they can be. The reason for having a separate table is that as systems become more complex, there may be more than one configuration table that points to these descriptions. Another reason for two tables is that this in not reproduced for every template and instance, which can save some additional data movement. Every designer will have to evaluate the tradeoffs between number of objects and data movement efficiency just as with other MIB modules.

Step 3: The bldgHVACCfgTemplateTable contains the specific configuration parameters that are pointed to by the bldgHVACConfigPtr object. Note that many rows in the bldgHVACTable can point to an entry in this table. It is also possible for entries to be used by 1 or 0 rows of the bldgHVACTable. It is the property of allowing multiple rows (instances) in the bldgHVACTable to point to a row in this table that can produce such efficiency gains from policy-based management with SNMP. Also notice that the configuration data is tied directly to the counter data so that people can see how configurations impact behavior.

The objects in this table are all that are necessary for configuration and connection to the other tables as well as the usual SnmpAdminString, StorageType, and RowStatus objects:

A simple index to the table:

bldgHVACCfgTemplateIndex

The configuration objects:

```
      bldgHVACCfgTemplateDesiredTemp
      bldgHVACCfgTemplateCoolOrHeat


Administrative objects for SnmpAdminString and
RowStatus:

       bldgHVACCfgTemplateInfo
       bldgHVACCfgTemplateOwner
       bldgHVACCfgTemplateStorage
       bldgHVACCfgTemplateStatus
```

8.1. MIB Module Definition

```
BLDG-HVAC-MIB DEFINITIONS ::= BEGIN
IMPORTS
    MODULE-IDENTITY, Counter32,
    Gauge32, OBJECT-TYPE, Unsigned32, experimental
        FROM SNMPv2-SMI
    MODULE-COMPLIANCE, OBJECT-GROUP
       FROM SNMPv2-CONF
    TEXTUAL-CONVENTION,
    TimeStamp, RowStatus, StorageType
        FROM SNMPv2-TC
    SnmpAdminString
        FROM SNMP-FRAMEWORK-MIB;


bldgHVACMIB MODULE-IDENTITY
    LAST-UPDATED "200303270000Z"
    ORGANIZATION "SNMPCONF working group
                  E-mail: snmpconf@snmp.com"
    CONTACT-INFO
        "Jon Saperia
        Postal:     JDS Consulting
                    174 Chapman Street
                    Watertown, MA 02472
                    U.S.A.
        Phone:      +1 617 744 1079
        E-mail:     saperia@jdscons.com

        Wayne Tackabury
        Postal:     Gold Wire Technology
                    411 Waverley Oaks Rd.
                    Waltham, MA 02452
                    U.S.A.
        Phone:      +1 781 398 8800
        E-mail:     wayne@goldwiretech.com
```

```
        Michael MacFaden
        Postal:     Riverstone Networks
                    5200 Great America Pkwy.
                    Santa Clara, CA 95054
                    U.S.A.
        Phone:      +1 408 878 6500
        E-mail:     mrm@riverstonenet.com

        David Partain
        Postal:     Ericsson AB
                    P.O. Box 1248
                    SE-581 12  Linkoping
                    Sweden
        E-mail:     David.Partain@ericsson.com"
    DESCRIPTION
        "This example MIB module defines a set of management objects
        for heating ventilation and air conditioning systems.  It
        also includes objects that can be used to create policies
        that are applied to rooms.  This eliminates the need to send
        per-instance configuration commands to the system.

        Copyright (C) The Internet Society (2003).  This version of
        this MIB module is part of RFC 3512; see the RFC itself for
        full legal notices."

    REVISION "200303270000Z"
    DESCRIPTION
        "Initial version of BLDG-HVAC-MIB as published in RFC 3512."
    ::= { experimental 122 }

bldgHVACObjects         OBJECT IDENTIFIER ::= { bldgHVACMIB 1 }
bldgConformance         OBJECT IDENTIFIER ::= { bldgHVACMIB 2 }

--
-- Textual Conventions
--

BldgHvacOperation  ::= TEXTUAL-CONVENTION
    STATUS             current
    DESCRIPTION
        "Operations supported by a heating and cooling system.
        A reference to underlying general systems would go here."
    SYNTAX      INTEGER {
                         heat(1),
                         cool(2)
                }
--
-- HVAC Objects Group
```

```
--

bldgHVACTable    OBJECT-TYPE
    SYNTAX      SEQUENCE OF BldgHVACEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "This table is the representation and data control
        for building HVAC by each individual office.
        The table has rows for, and is indexed by a specific
        floor and office number.  Each such row includes
        HVAC statistical and current status information for
        the associated office.  The row also contains a
        bldgHVACCfgTemplate columnar object that relates the
        bldgHVACTable row to a row in the bldgHVACCfgTemplateTable.
        If this value is nonzero, then the instance in the row
        that has a value for how the HVAC has been configured
        in the associated template (bldgHVACCfgTeplateTable row).
        Hence, the bldgHVACCfgTeplateTable row contains the
        specific configuration values for the offices as described
        in this table."
    ::= { bldgHVACObjects 1 }

bldgHVACEntry  OBJECT-TYPE
    SYNTAX       BldgHVACEntry
    MAX-ACCESS   not-accessible
    STATUS       current
    DESCRIPTION
        "A row in the bldgHVACTable.  Each row represents a particular
        office in the building, qualified by its floor and office
        number.  A given row instance can be created or deleted by
        set operations  upon its bldgHVACStatus columnar
        object instance."
    INDEX { bldgHVACFloor, bldgHVACOffice }
        ::= { bldgHVACTable 1 }

BldgHVACEntry ::= SEQUENCE {
        bldgHVACFloor             Unsigned32,
        bldgHVACOffice            Unsigned32,
        bldgHVACCfgTemplate       Unsigned32,
        bldgHVACFanSpeed          Gauge32,
        bldgHVACCurrentTemp       Gauge32,
        bldgHVACCoolOrHeatMins    Counter32,
        bldgHVACDiscontinuityTime TimeStamp,
        bldgHVACOwner             SnmpAdminString,
        bldgHVACStorageType       StorageType,
        bldgHVACStatus            RowStatus
        }
```

```
bldgHVACFloor    OBJECT-TYPE
    SYNTAX      Unsigned32 (1..1000)
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "This portion of the index indicates the floor of the
         building.  The ground floor is considered the
         first floor.  For the purposes of this example,
         floors under the ground floor cannot be
         controlled using this MIB module."
    ::= { bldgHVACEntry 1 }

bldgHVACOffice    OBJECT-TYPE
    SYNTAX      Unsigned32 (1..2147483647)
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "This second component of the index specifies the
        office number."
    ::= { bldgHVACEntry 2 }

bldgHVACCfgTemplate  OBJECT-TYPE
    SYNTAX          Unsigned32
    MAX-ACCESS      read-create
    STATUS          current
    DESCRIPTION
        "The index (bldgHVACCfgTemplateIndex instance)
        of an entry in the 'bldgHVACCfgTemplateTable'.
        The bldgHVACCfgTable row instance referenced
        is a pre-made configuration 'template'
        that represents the configuration described
        by the bldgHVACCfgTemplateInfoDescr object.  Note
        that not all configurations will be under a
        defined template.  As a result, a row in this
        bldgHVACTable may point to an entry in the
        bldgHVACCfgTemplateTable that does not in turn
        have a reference (bldgHVACCfgTemplateInfo) to an
        entry in the bldgHVACCfgTemplateInfoTable.  The
        benefit of this approach is that all
        configuration information is available in one
        table whether all elements in the system are
        derived from configured templates or not.

        Where the instance value for this colunmar object
        is zero, this row represents data for an office
        whose HVAC status can be monitored using the
        read-only columnar object instances of this
        row, but is not under the configuration control
```

```
        of the agent."
    ::= { bldgHVACEntry 3 }

bldgHVACFanSpeed  OBJECT-TYPE
    SYNTAX            Gauge32
    UNITS             "revolutions per minute"
    MAX-ACCESS        read-only
    STATUS            current
    DESCRIPTION
        "Shows the revolutions per minute of the fan.  Fan speed
        will vary based on the difference between
        bldgHVACCfgTemplateDesiredTemp and bldgHVACCurrentTemp.  The
        speed is measured in revolutions of the fan blade per minute."
    ::= { bldgHVACEntry 4 }

bldgHVACCurrentTemp  OBJECT-TYPE
    SYNTAX            Gauge32
    UNITS             "degrees in celsius"
    MAX-ACCESS        read-only
    STATUS            current
    DESCRIPTION
        "The current measured temperature in the office.  Should
        the current temperature be measured at a value of less
        than zero degrees celsius, a read of the instance
        for this object will return a value of zero."
    ::= { bldgHVACEntry 5 }

bldgHVACCoolOrHeatMins  OBJECT-TYPE
    SYNTAX            Counter32
    UNITS             "minutes"
    MAX-ACCESS        read-only
    STATUS            current
    DESCRIPTION
        "The total number of heating or cooling minutes that have
        been consumed since the row was activated.  Notice that
        whether the minutes represent heating or cooling is a
        function of the configuration of this row.  If the system
        is re-initialized from a cooling to heating function or
        vice versa, then the counter would start over again.  This
        effect is similar to a reconfiguration of some network
        interface cards.  When parameters that impact
        configuration are changed, the subsystem must be
        re-initialized.  Discontinuities in the value of this counter
        can occur at re-initialization of the management system,
        and at other times as indicated by the value of
        bldgHVACDiscontinuityTime."
    ::= { bldgHVACEntry 6 }
```

```
bldgHVACDiscontinuityTime OBJECT-TYPE
    SYNTAX      TimeStamp
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The value of sysUpTime on the most recent occasion at which
        any heating or cooling operation for the office designated
        by this row instance experienced a discontinuity.  If
        no such discontinuities have occurred since the last re-
        initialization of the this row, then this object contains a
        zero value."
    ::= { bldgHVACEntry 7 }

bldgHVACOwner  OBJECT-TYPE
    SYNTAX            SnmpAdminString
    MAX-ACCESS        read-create
    STATUS            current
    DESCRIPTION
        "The identity of the operator/system that
        last modified this entry.  When a new entry
        is created, a valid SnmpAdminString must
        be supplied.  If, on the other hand, this
        entry is populated by the agent 'discovering'
        unconfigured rooms, the empty string is a valid
        value for this object."
    ::= { bldgHVACEntry 8 }

bldgHVACStorageType  OBJECT-TYPE
    SYNTAX            StorageType
    MAX-ACCESS        read-create
    STATUS            current
    DESCRIPTION
        "The persistence of this row of the table in system storage,
        as it pertains to permanence across system resets.  A columnar
        instance of this object with value 'permanent' need not allow
        write-access to any of the columnar object instances in the
        containing row."
    ::= { bldgHVACEntry 9  }

bldgHVACStatus  OBJECT-TYPE
    SYNTAX            RowStatus
    MAX-ACCESS        read-create
    STATUS            current
    DESCRIPTION
        "Controls and reflects the creation and activation status of
        a row in this table.

        No attempt to modify a row columnar object instance value in
```

```
        the bldgHVACTable should be issued while the value of
        bldgHVACStatus is active(1).  Should an agent receive a SET
        PDU attempting such a modification in this state, an
        inconsistentValue error should be returned as a result of
        the SET attempt."
    ::= { bldgHVACEntry 10 }
--
-- HVAC Configuration Template Table
--

bldgHVACCfgTemplateInfoTable  OBJECT-TYPE
    SYNTAX      SEQUENCE OF BldgHVACCfgTemplateInfoEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "This table provides unique string identification for
        HVAC templates in a network.  If it were necessary to
        configure rooms to deliver a particular quality of climate
        control with regard to cooling or heating, the index string
        of a row in this table could be the template name.
        The bldgHVACCfgCfgTemplateInfoDescription
        contains a brief description of the template service objective
        such as: provides summer cooling settings for executive
        offices.  The bldgHVACCfgTemplateInfo in the
        bldgHVACCfgTemplateTable will contain the pointer to the
        relevant row in this table if it is intended that items
        that point to a row in the bldgHVACCfgTemplateInfoTable be
        identifiable as being under template control though this
        mechanism."

    ::= { bldgHVACObjects 2 }

bldgHVACCfgTemplateInfoEntry  OBJECT-TYPE
    SYNTAX       BldgHVACCfgTemplateInfoEntry
    MAX-ACCESS   not-accessible
    STATUS       current
    DESCRIPTION
        "Each row represents a particular template and
        description.  A given row instance can be created or
        deleted by set operations upon its
        bldgHVACCfgTemplateInfoStatus columnar object
        instance."
    INDEX { bldgHVACCfgTemplateInfoIndex }
    ::= { bldgHVACCfgTemplateInfoTable 1 }

BldgHVACCfgTemplateInfoEntry ::= SEQUENCE {
        bldgHVACCfgTemplateInfoIndex          Unsigned32,
        bldgHVACCfgTemplateInfoID             SnmpAdminString,
```

```
        bldgHVACCfgTemplateInfoDescr          SnmpAdminString,
        bldgHVACCfgTemplateInfoOwner          SnmpAdminString,
        bldgHVACCfgTemplateInfoStatus         RowStatus,
        bldgHVACCfgTemplateInfoStorType       StorageType
        }

bldgHVACCfgTemplateInfoIndex   OBJECT-TYPE
       SYNTAX       Unsigned32 (1..2147483647)
       MAX-ACCESS   not-accessible
       STATUS       current
       DESCRIPTION
           "The unique index to a row in this table."
        ::= { bldgHVACCfgTemplateInfoEntry 1 }

bldgHVACCfgTemplateInfoID  OBJECT-TYPE
    SYNTAX       SnmpAdminString
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION
        "Textual identifier for this table row, and, consequently
        the template.  This should be a unique name within
        an administrative domain for a particular template so that
        all systems in a network that are under the same template
        can have the same 'handle' (e.g., 'Executive Offices',
        'Lobby Areas')."
    ::= { bldgHVACCfgTemplateInfoEntry 2 }


bldgHVACCfgTemplateInfoDescr   OBJECT-TYPE
    SYNTAX       SnmpAdminString
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION
        "A general description of the template.  One example might
        be - Controls the cooling for offices on higher floors
        during the summer."
    ::= { bldgHVACCfgTemplateInfoEntry 3 }

bldgHVACCfgTemplateInfoOwner  OBJECT-TYPE
    SYNTAX            SnmpAdminString
    MAX-ACCESS        read-create
    STATUS            current
    DESCRIPTION
        "The identity of the operator/system that last modified
        this entry."
    ::= { bldgHVACCfgTemplateInfoEntry 4 }

bldgHVACCfgTemplateInfoStatus  OBJECT-TYPE
```

```
    SYNTAX            RowStatus
    MAX-ACCESS        read-create
    STATUS            current
    DESCRIPTION
        "The activation status of this row.

        No attempt to modify a row columnar object instance value in
        the bldgHVACCfgTemplateInfo Table should be issued while the
        value of bldgHVACCfgTemplateInfoStatus is active(1).
        Should an agent receive a SET PDU attempting such a modification
        in this state, an inconsistentValue error should be returned as
        a result of the SET attempt."
    ::= { bldgHVACCfgTemplateInfoEntry 5 }

bldgHVACCfgTemplateInfoStorType   OBJECT-TYPE
    SYNTAX            StorageType
    MAX-ACCESS        read-create
    STATUS            current
    DESCRIPTION
        "The persistence of this row of the table in system storage,
         as it pertains to permanence across system resets.  A columnar
        instance of this object with value 'permanent' need not allow
        write-access to any of the columnar object instances in the
        containing row."
    ::= { bldgHVACCfgTemplateInfoEntry 6  }

--
-- HVAC Configuration Template Table
--
bldgHVACCfgTemplateTable    OBJECT-TYPE
    SYNTAX      SEQUENCE OF BldgHVACCfgTemplateEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "This table contains the templates, which
        can be used to set defaults that will
        be applied to specific offices.  The application
        of those values is accomplished by having a row
        instance of the bldgHVACTable reference a row of
        this table (by the value of the former's
        bldgHVACCfgTemplate columnar instance).  Identifying
        information concerning a row instance of this table
        can be found in the columnar data of the row instance
        of the bldgHVACCfgTemplateInfoTable entry referenced
        by the bldgHVACCfgTemplateInfo columnar object of
        this table."
    ::= { bldgHVACObjects 3 }
```

```
bldgHVACCfgTemplateEntry  OBJECT-TYPE
    SYNTAX       BldgHVACCfgTemplateEntry
    MAX-ACCESS   not-accessible
    STATUS       current
    DESCRIPTION
        "Each row represents a single set of template parameters
        that can be applied to selected instances - in this case
        offices.  These policies will be turned on and off by the
        policy module through its scheduling facilities.

        A given row instance can be created or
        deleted by set operations upon its
        bldgHVACCfgTemplateStatus columnar object instance."
    INDEX { bldgHVACCfgTemplateIndex }
    ::= { bldgHVACCfgTemplateTable 1 }

BldgHVACCfgTemplateEntry ::= SEQUENCE {
        bldgHVACCfgTemplateIndex           Unsigned32,
        bldgHVACCfgTemplateDesiredTemp     Gauge32,
        bldgHVACCfgTemplateCoolOrHeat      BldgHvacOperation,
        bldgHVACCfgTemplateInfo            Unsigned32,
        bldgHVACCfgTemplateOwner           SnmpAdminString,
        bldgHVACCfgTemplateStorage         StorageType,
        bldgHVACCfgTemplateStatus          RowStatus
}

bldgHVACCfgTemplateIndex    OBJECT-TYPE
    SYNTAX      Unsigned32 (1..2147483647)
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "A unique value for each defined template in this
        table.  This value can be referenced as a row index
        by any MIB module that needs access to this information.
        The bldgHVACCfgTemplate will point to entries in this
        table."
    ::= { bldgHVACCfgTemplateEntry 1 }

bldgHVACCfgTemplateDesiredTemp OBJECT-TYPE
    SYNTAX            Gauge32
    UNITS             "degrees in celsius"
    MAX-ACCESS        read-create
    STATUS            current
    DESCRIPTION
        "This is the desired temperature setting.  It might be
        changed at different times of the day or based on
        seasonal conditions.  It is permitted to change this value
        by first moving the row to an inactive state, making the
```

```
        change and then reactivating the row."
    ::= { bldgHVACCfgTemplateEntry 2 }

bldgHVACCfgTemplateCoolOrHeat  OBJECT-TYPE
    SYNTAX             BldgHvacOperation
    MAX-ACCESS         read-create
    STATUS             current
    DESCRIPTION
        "This controls the heating and cooling mechanism and is
        set-able by building maintenance.  It is permitted to
        change this value by first moving the row to an inactive
        state, making the change and then reactivating the row."
    ::= { bldgHVACCfgTemplateEntry 3 }

bldgHVACCfgTemplateInfo OBJECT-TYPE
    SYNTAX             Unsigned32
    MAX-ACCESS         read-create
    STATUS             current
    DESCRIPTION
        "This object points to a row in the
        bldgHVACCfgTemplateInfoTable.  This controls the
        heating and cooling mechanism and is set-able by
        building maintenance.  It is permissible to change
        this value by first moving the row to an inactive
        state, making the change and then reactivating
        the row.  A value of zero means that this entry
        is not associated with a named template found
        in the bldgHVACCfgTemplateInfoTable."
    ::= { bldgHVACCfgTemplateEntry 4 }

bldgHVACCfgTemplateOwner  OBJECT-TYPE
    SYNTAX            SnmpAdminString
    MAX-ACCESS        read-create
    STATUS            current
    DESCRIPTION
        "The identity of the administrative entity
        that created this row of the table."
    ::= { bldgHVACCfgTemplateEntry 5 }

bldgHVACCfgTemplateStorage  OBJECT-TYPE
    SYNTAX            StorageType
    MAX-ACCESS        read-create
    STATUS            current
    DESCRIPTION
        "The persistence of this row of the table across
         system resets.  A columnar instance of this object with
         value 'permanent' need not allow write-access to any
         of the columnar object instances in the containing row."
```

```
    ::= { bldgHVACCfgTemplateEntry 6 }

bldgHVACCfgTemplateStatus  OBJECT-TYPE
    SYNTAX            RowStatus
    MAX-ACCESS        read-create
    STATUS            current
    DESCRIPTION
        "The activation status of this row of the table.

        No attempt to modify a row columnar object instance value in
        the bldgHVACCfgTemplateTable should be issued while the
        value of bldgHVACCfgTemplateStatus is active(1).
        Should an agent receive a SET PDU attempting such a modification
        in this state, an inconsistentValue error should be returned as
        a result of the SET attempt."
    ::= { bldgHVACCfgTemplateEntry 7 }

--
-- Conformance Information
--

bldgCompliances  OBJECT IDENTIFIER ::= { bldgConformance 1 }
bldgGroups       OBJECT IDENTIFIER ::= { bldgConformance 2 }

-- Compliance Statements

bldgCompliance MODULE-COMPLIANCE
    STATUS current
    DESCRIPTION
        "The requirements for conformance to the BLDG-HVAC-MIB.  The
         bldgHVACObjects group must be implemented to conform to the
         BLDG-HVAC-MIB."
    MODULE -- this module

    GROUP bldgHVACObjectsGroup
    DESCRIPTION
        "The bldgHVACObjects is mandatory for all systems that
        support HVAC systems."
    ::= { bldgCompliances 1 }

bldgHVACObjectsGroup OBJECT-GROUP
    OBJECTS {
        bldgHVACCfgTemplate,
        bldgHVACFanSpeed, bldgHVACCurrentTemp,
        bldgHVACCoolOrHeatMins, bldgHVACDiscontinuityTime,
        bldgHVACOwner, bldgHVACStatus,
        bldgHVACStorageType, bldgHVACCfgTemplateInfoID,
        bldgHVACCfgTemplateInfoDescr, bldgHVACCfgTemplateInfoOwner,
```

```
        bldgHVACCfgTemplateInfoStatus,
        bldgHVACCfgTemplateInfoStorType,
        bldgHVACCfgTemplateDesiredTemp,
        bldgHVACCfgTemplateCoolOrHeat,
        bldgHVACCfgTemplateInfo,
        bldgHVACCfgTemplateOwner,bldgHVACCfgTemplateStorage,
        bldgHVACCfgTemplateStatus
    }
    STATUS current
    DESCRIPTION
        "The bldgHVACObjects Group."
    ::= { bldgGroups 1 }

END
```

8.2. Notes on MIB Module with Template-based Data

The primary purpose of the example "HVAC" MIB module is to show how to construct a single module that includes configuration, template, counter and state information in a single module. If this were a 'real' module we would also have included definitions for notifications for the configuration change operations as previously described. We also would have included notifications for faults and other counter threshold events.

Implementation and Instance Extensions:

Just as with networking technologies, vendors may wish to add extensions that can distinguish their products from the competition. If an HVAC vendor also wanted to support humidity control, they could add that facility to their equipment and use AUGMENTS for the bldgHVACTemplateTable with two objects, one that indicates the desired humidity and the other, the actual. The bldgHVACTemplateTable could also be extended using this same approach so that HVAC policies could easily be extended to support this vendor.

8.3. Examples of Usage of the MIB

The following two examples use two templates to configure the temperature in executive offices and in conference rooms. The "conference rooms" template is applied to all conference rooms (which happen to be office 104 on each floor), and the "executive offices" template is applied to executive offices.

If offices 24, 25, and 26 on the third floor are executive offices, the values in the bldgHVACTable might be:

```
bldgHVACCfgTemplate.3.24 = 2
bldgHVACFanSpeed.3.24 = 2989
bldgHVACCurrentTemp.3.24 = 24
bldgHVACCoolOrHeatMins.3.24 = 123
bldgHVACDiscontinuityTime.3.24 = sysUpTime + 12h + 21m
bldgHVACOwner.3.24 = "policy engine"
bldgHVACStorageType.3.24 = nonVolatile(3)
bldgHVACStatus.3.24 = active(1)

bldgHVACCfgTemplate.3.25 = 2
bldgHVACFanSpeed.3.25 = 0
bldgHVACCurrentTemp.3.25 = 22
bldgHVACCoolOrHeatMins.3.25 = 298
bldgHVACDiscontinuityTime.3.25 = sysUpTime + 4h + 2m
bldgHVACOwner.3.25 = "policy engine"
bldgHVACStorageType.3.25 = nonVolatile(3)
bldgHVACStatus.3.25 = active(1)

bldgHVACCfgTemplate.3.26 = 2
bldgHVACFanSpeed.3.26 = 0
bldgHVACCurrentTemp.3.26 = 22
bldgHVACCoolOrHeatMins.3.26 = 982
bldgHVACOwner.3.26 = "policy engine"
bldgHVACStorageType.3.26 = nonVolatile(3)
bldgHVACStatus.3.26 = active(1)
```

The second entry in the bldgHVACCfgTemplateTable, to which all of the above point, might have the following configuration:

```
bldgHVACCfgTemplateDesiredTemp.2 = 22
bldgHVACCfgTemplateCoolOrHeat.2 = cool(2)
bldgHVACCfgTemplateInfo.2 = 2
bldgHVACCfgTemplateOwner.2 = "Senior Executive assistant"
bldgHVACCfgTemplateStorage.2 = nonVolatile(3)
bldgHVACCfgTemplateStatus.2 = active(1)
```

and the associated template information ("executive offices") might be:

```
bldgHVACCfgTemplateInfoID.2 = "executive offices"
bldgHVACCfgTemplateInfoDescr.2 = "Controls temperature for executive
                                  offices"
bldgHVACCfgTemplateInfoOwner.2 = "Senior Executive assistant"
bldgHVACCfgTemplateInfoStorType.2 = nonVolatile(3)
bldgHVACCfgTemplateInfoStatus.2 = active(1)
```

The policy engine can now associate instances of executive offices with the template called "executive offices" and apply the values in the second entry of the bldgHVACCfgTemplateTable to each of the instances of the executive offices. This will then attempt to set the temperature in executive offices to 22 degrees celsius.

It is also possible that there may be an office configured for a particular temperature, but without using a template. For example, office 28 on the third floor might look like this:

```
bldgHVACCfgTemplate.3.28 = 3
bldgHVACFanSpeed.3.28 = 50
bldgHVACCurrentTemp.3.28 = 26
bldgHVACCoolOrHeatMins.3.28 = 0
bldgHVACDiscontinuityTime.3.28 = 0
bldgHVACOwner.3.28 = "Executive with poor circulation"
bldgHVACStorageType.3.28 = nonVolatile(3)
bldgHVACStatus.3.28 = active(1)
```

The entry in the bldgHVACCfgTemplateTable (to which bldgHVACCfgTemplate.3.28 points) might instead look like:

```
bldgHVACCfgTemplateDesiredTemp.3 = 28
bldgHVACCfgTemplateCoolOrHeat.3 = cool(2)
bldgHVACCfgTemplateInfo.3 = 0.0
bldgHVACCfgTemplateOwner.3 = "Executive with poor circulation"
bldgHVACCfgTemplateStorage.3 = nonVolatile(3)
bldgHVACCfgTemplateStatus.3 = active(1)
```

Note that this entry does not point to a template.

If the executive’s circulation improves so that the temperature should be aligned with other executive offices, this is accomplished by changing the value of bldgHVACCfgTemplate.3.28 from bldgHVACCfgTemplateInfoID.3 to bldgHVACCfgTemplateInfoID.2 (shown above).

Finally, there might be offices for which there is no configured temperature but management applications can read the current temperature, fan speed, and cooling or heating minutes from the bldgHVACTable. In that case, the value of bldgHVACCfgTemplate will be a zero index ("null"), as will the value of bldgHVACOwner.

```
bldgHVACCfgTemplate.4.2 = 0
bldgHVACFanSpeed.3.28 = 50
bldgHVACCurrentTemp.3.28 = 26
bldgHVACCoolOrHeatMins.3.28 = 0
bldgHVACDiscontinuityTime.3.28 = 0
bldgHVACOwner.3.28 = ""
bldgHVACStorageType.3.28 = nonVolatile(3)
bldgHVACStatus.3.28 = active(1)
```

As a second example, the conference rooms on several floors are configured using the "conference rooms" template. When the values in the bldgHVACTable pertaining to conference rooms are read, it might look like:

```
bldgHVACCfgTemplate.12.104 = bldgHVACCfgTemplateDesiredTemp.1
bldgHVACFanSpeed.12.104 = 1423
bldgHVACCurrentTemp.12.104 = 21
bldgHVACCoolOrHeatMins.12.104 = 2193
bldgHVACDiscontinuityTime.12.104 = sysUpTime + 36h + 15m
bldgHVACOwner.12.104 =  = "Bob the Conference Guy"
bldgHVACStorageType.12.104 = nonVolatile(3)
bldgHVACStatus.12.104 = active(1)

bldgHVACCfgTemplate.14.104 = bldgHVACCfgTemplateDesiredTemp.1
bldgHVACFanSpeed.14.104 = 1203
bldgHVACCurrentTemp.14.104 = 20
bldgHVACCoolOrHeatMins.14.104 = 293
bldgHVACDiscontinuityTime.14.104 = sysUpTime + 5h + 54m
bldgHVACOwner.14.104 =  = "Bob the Conference Guy"
bldgHVACStorageType.14.104 = nonVolatile(3)
bldgHVACStatus.14.104 = active(1)

bldgHVACCfgTemplate.15.104 = bldgHVACCfgTemplateDesiredTemp.1
bldgHVACFanSpeed.15.104 = 12
bldgHVACCurrentTemp.15.104 = 19
bldgHVACCoolOrHeatMins.15.104 = 1123
bldgHVACDiscontinuityTime.15.103 = sysUpTime + 2d + 2h + 7m
bldgHVACOwner.15.104 =  = "Bob the Conference Guy"
bldgHVACStorageType.15.104 = nonVolatile(3)
bldgHVACStatus.15.104 = active(1)
```

The desired temperature and whether to heat or cool is configured in the first entry of the bldgHVACCfgTemplateTable, which tries to set the temperature to 19 degrees celsius in conference rooms:

```
bldgHVACCfgTemplateDesiredTemp.1 = 19
bldgHVACCfgTemplateCoolOrHeat.1 = cool(2)
bldgHVACCfgTemplateInfo.1 = bldgHVACCfgTemplateInfoID.1
bldgHVACCfgTemplateOwner.1 = "Bob the Conference Guy"
bldgHVACCfgTemplateStorage.1 = nonVolatile(3)
bldgHVACCfgTemplateStatus.1 = active(1)
```

The associated template information would then have:

```
bldgHVACCfgTemplateInfoID.1 = "conference rooms"
bldgHVACCfgTemplateInfoDescr.1 = "Controls temperature in conference
rooms" bldgHVACCfgTemplateInfoOwner.1 = "Bob the Conference Guy"
bldgHVACCfgTemplateInfoStorType.1 = nonVolatile(3)
bldgHVACCfgTemplateInfoStatus.1 = active(1)
```

The policy system can then apply this template (cool to 19 degrees Celsius) to its notion of all of the conference rooms in the building.

9. Security Considerations

This document discusses practices and methods for using the SNMP for management and distribution of configuration information for network elements. Any effective use of the SNMP in this application must concern itself with issues of authentication of the management entities initiating configuration change and management, in addition to the integrity of the configuration data itself. Other more subtle considerations also exist.

To that end, the section of this document entitled "Deployment and Security Issues" covers these security considerations to the extent they affect the current practices described throughout this document. In particular, in the subsection entitled "Secure Agent Considerations", there is a recommendation for the usage of Version 3 of the SNMP, and its essential presumption as a foundation for other practices described throughout. With the exception of a small number of cases where a mention is made to the contrary to illustrate techniques for coexistence with application entities dependent upon earlier versions of the SNMP, that recommendation of usage of Version 3 of the SNMP is reiterated here.

10. Acknowledgments

This document was produced by the SNMPCONF Working Group. In particular, the editors wish to thank:

Christopher Anderson
Andy Bierman
Greg Bruell
Dr Jeffrey Case
Chris Elliott
Joel Halpern
Pablo Halpern
Wes Hardaker
David Harrington
Harrie Hazewinkel
Thippanna Hongal
Bob Moore
David T. Perkins
Randy Presuhn
Dan Romascanu
Shawn Routhier
Steve Waldbusser
Bert Wijnen

11. Normative References

[1] Harrington, D., Presuhn, R. and B. Wijnen, "An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks", STD 62, RFC 3411, December 2002.

[2] McCloghrie, K., Perkins, D. and J. Schoenwaelder, "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, April 1999.

[3] McCloghrie, K., Perkins, D. and J. Schoenwaelder, "Textual Conventions for SMIv2", STD 58, RFC 2579, April 1999.

[4] McCloghrie, K., Perkins, D. and J. Schoenwaelder, "Conformance Statements for SMIv2", STD 58, RFC 2580, April 1999.

[5] Presuhn, R. (Ed.), "Transport Mappings for the Simple Network Management Protocol (SNMPv2)", STD 62, RFC 3417, December 2002.

[6] Case, J., Harrington D., Presuhn R. and B. Wijnen, "Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3412, December 2002.

[7] Blumenthal, U. and B. Wijnen, "User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)", STD 62, RFC 3414, December 2002.

[8] Presuhn, R. (Ed.), "Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3416, December 2002.

[9] Levi, D., Meyer, P., and B. Stewart, "Simple Network Management Protocol Applications", STD 62, RFC 3413, December 2002.

[10] Wijnen, B., Presuhn, R. and K. McCloghrie, "View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3415, December 2002.

[11] Presuhn, R. (Ed.), "Management Information Base for the Simple Network Management Protocol (SNMPv2)", STD 62, RFC 3418, December 2002.

[12] Case, J., Mundy, R., Partain, D. and B. Stewart, "Introduction and Applicability Statements for Internet-Standard Management Framework", RFC 3410, December 2002.

[13] Daniele, M., Haberman, B., Routhier, S. and J. Schoenwaelder, "Textual Conventions for Internet Network Addresses", RFC 3291, May 2002.

[14] McCloghrie, K. (Ed.), "SNMPv2 Management Information Base for the Internet Protocol using SMIv2", RFC 2011, November 1996.

12. Informative References

[15] Rose, M. and K. McCloghrie, "Structure and Identification of Management Information for TCP/IP-based Internets", STD 16, RFC 1155, May 1990.

[16] Rose, M. and K. McCloghrie, "Concise MIB Definitions", STD 16, RFC 1212, March 1991.

[17] Rose, M., "A Convention for Defining Traps for use with the SNMP", RFC 1215, March 1991.

[18] Case, J., Fedor, M., Schoffstall, M. and J. Davin, "Simple Network Management Protocol", STD 15, RFC 1157, May 1990.

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

[20] McCloghrie, K. and F. Kastenholz, "The Interfaces Group MIB", RFC 2863, June 2000.

[21] Brown, C. and F. Baker, "Management Information Base for Frame Relay DTEs Using SMIv2", RFC 2115, September 1997.

[22] Baker, F. (Ed.), "Requirements for IP Version 4 Routers", RFC 1812, June 1995.

[23] Hawkinson, J. and T. Bates, "Guidelines for Creation, Selection, and Registration of an Autonomous System (AS)", BCP 6, RFC 1930, March 1996.

[24] Decker, E., Langille, P., Rijsinghani, A. and K. McCloghrie, "Definitions of Managed Objects for Bridges", RFC 1493, July 1993.

[25] Levi, D. and J. Schoenwaelder "Definitions of Managed Objects for Scheduling Management Operations", RFC 3231, January 2002.

[26] Bell, E., Smith, A., Langille, P., Rijsinghani, A. and K. McCloghrie, "Definitions of Managed Objects for Bridges with Traffic Classes, Multicast Filtering and Virtual LAN Extensions", RFC 2674, August 1999.

[27] Baker, F., "IP Forwarding Table MIB", RFC 2096, January 1997.

[28] St. Johns, M. (Ed.), "Radio Frequency (RF) Interface Management Information Base for MCNS/DOCSIS compliant RF interfaces", RFC 2670, August 1999.

[29] Baker, F. and R. Coltun, "OSPF Version 2 Management Information Base", RFC 1850, November 1995.

[30] Blake, S., Black, D., Carlson M., Davies, E., Wang, Z. and W. Weiss, "An Architecture for Differentiated Services ", RFC 2475, December 1998.

[31] Willis, S., Burruss, J. and J. Chu (Ed.), "Definitions of Managed Objects for the Fourth Version of the Border Gateway Protocol (BGP-4) using SMIv2", RFC 1657, July 1994.

[32] Waldbusser, S., "Remote Network Monitoring Management Information Base", RFC 2819, May 2000.

[33] McCloghrie, K. and G. Hanson, "The Inverted Stack Table Extension to the Interfaces Group MIB", RFC 2864, June 2000.

[34] McCloghrie, K. and A. Bierman, "Entity MIB (Version 2)", RFC 2737, December 1999.

[35] ITU-T,, Recommendation M.3010., PRINCIPLES FOR A TELECOMMUNICATIONS MANAGEMENT NETWORK. February, 2000.

[36] Waldbusser, S., Saperia, J., and Hongal, T., "Policy Based Management MIB", Work-in-progress.

[37] Heintz, L., "SNMP Row Operations Extensions", Work-in-progress.

[38] Zeltserman, D., "A Practical Guide to Snmpv3 and Network Management", Prentice Hall, 1999.

[39] Noto, M., Spiegel, E. and K. Tesink, "Definitions of Textual Conventions and OBJECT-IDENTITIES for ATM Management", RFC 2514, February 1999.

[40] Kassaveri, R., Editor, "Distributed Management Expression MIB", RFC 2982, October 2000.

[41] St. Johns, M., "DOCSIS Cable Device MIB Cable Device Management Information Base for DOCSIS compliant Cable Modems and Cable Modem Termination Systems", RFC 2669, August 1999.

[42] Westerinen, A., Schnizlein, J., Strassner, J., Scherling, M., Quinn, B., Herzog, S., Huynh, A., Carlson, M., Perry, J. and S. Waldbusser, "Terminology for Policy-Based Management", RFC 3198, November 2001.

[43] http://wwww.cisco.com/univercd/cc/td/product/software/ios113ed/11ed_cr/secur_c/scprt/scacls.pdf.

[44] Waldbusser, S., "Remote Network Monitoring Management Information Base Version 2 using SMIv2", RFC 2021, January 1997.

13. Intellectual Property

obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification can be obtained from the IETF Secretariat.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this standard. Please address the information to the IETF Executive Director.

14. Editors' Addresses

Michael R. MacFaden
Riverstone Networks, Inc
5200 Great America Parkway
Santa Clara, CA 95054

EMail: mrm@riverstonenet.com

David Partain
Ericsson AB
P.O. Box 1248
SE-581 12 Linkoping
Sweden

EMail: David.Partain@ericsson.com

Jon Saperia
JDS Consulting
174 Chapman Street
Watertown, MA 02472

EMail: saperia@jdscons.com

Wayne F. Tackabury
Gold Wire Technology
411 Waverley Oaks Rd.
Waltham, MA 02452

EMail: wayne@goldwiretech.com

15. Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.