Network Working Group
Request for Comments: 3411
STD: 62
Obsoletes: 2571
Category: Standards Track

D. Harrington
Enterasys Networks
R. Presuhn
BMC Software, Inc.
B. Wijnen
Lucent Technologies

December 2002

An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks

Status of this Memo

Copyright Notice

Abstract

This document describes an architecture for describing Simple Network Management Protocol (SNMP) Management Frameworks. The architecture is designed to be modular to allow the evolution of the SNMP protocol standards over time. The major portions of the architecture are an SNMP engine containing a Message Processing Subsystem, a Security Subsystem and an Access Control Subsystem, and possibly multiple SNMP applications which provide specific functional processing of management data. This document obsoletes RFC 2571.

Table of Contents

1. Introduction

1.1. Overview

This document defines a vocabulary for describing SNMP Management Frameworks, and an architecture for describing the major portions of SNMP Management Frameworks.

This document does not provide a general introduction to SNMP. Other documents and books can provide a much better introduction to SNMP. Nor does this document provide a history of SNMP. That also can be found in books and other documents.

Section 1 describes the purpose, goals, and design decisions of this architecture.

Section 2 describes various types of documents which define (elements of) SNMP Frameworks, and how they fit into this architecture. It also provides a minimal road map to the documents which have previously defined SNMP frameworks.

Section 3 details the vocabulary of this architecture and its pieces. This section is important for understanding the remaining sections, and for understanding documents which are written to fit within this architecture.

Section 4 describes the primitives used for the abstract service interfaces between the various subsystems, models and applications within this architecture.

Section 5 defines a collection of managed objects used to instrument SNMP entities within this architecture.

Sections 6, 7, 8, 9, 10 and 11 are administrative in nature.

Appendix A contains guidelines for designers of Models which are expected to fit within this architecture.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

1.2. SNMP

An SNMP management system contains:

- several (potentially many) nodes, each with an SNMP entity containing command responder and notification originator applications, which have access to management instrumentation (traditionally called agents);

- at least one SNMP entity containing command generator and/or notification receiver applications (traditionally called a manager) and,

- a management protocol, used to convey management information between the SNMP entities.

SNMP entities executing command generator and notification receiver applications monitor and control managed elements. Managed elements are devices such as hosts, routers, terminal servers, etc., which are monitored and controlled via access to their management information.

It is the purpose of this document to define an architecture which can evolve to realize effective management in a variety of configurations and environments. The architecture has been designed to meet the needs of implementations of:

- minimal SNMP entities with command responder and/or notification originator applications (traditionally called SNMP agents),

- SNMP entities with proxy forwarder applications (traditionally called SNMP proxy agents),

- command line driven SNMP entities with command generator and/or notification receiver applications (traditionally called SNMP command line managers),

- SNMP entities with command generator and/or notification receiver, plus command responder and/or notification originator applications (traditionally called SNMP mid-level managers or dual-role entities),

- SNMP entities with command generator and/or notification receiver and possibly other types of applications for managing a potentially very large number of managed nodes (traditionally called (network) management stations).

1.3. Goals of this Architecture

This architecture was driven by the following goals:

- Use existing materials as much as possible. It is heavily based on previous work, informally known as SNMPv2u and SNMPv2*, based in turn on SNMPv2p.

- Address the need for secure SET support, which is considered the most important deficiency in SNMPv1 and SNMPv2c.

- Make it possible to move portions of the architecture forward in the standards track, even if consensus has not been reached on all pieces.

- Define an architecture that allows for longevity of the SNMP Frameworks that have been and will be defined.

- Keep SNMP as simple as possible.

- Make it relatively inexpensive to deploy a minimal conforming implementation.

- Make it possible to upgrade portions of SNMP as new approaches become available, without disrupting an entire SNMP framework.

- Make it possible to support features required in large networks, but make the expense of supporting a feature directly related to the support of the feature.

1.4. Security Requirements of this Architecture

Several of the classical threats to network protocols are applicable to the management problem and therefore would be applicable to any Security Model used in an SNMP Management Framework. Other threats are not applicable to the management problem. This section discusses principal threats, secondary threats, and threats which are of lesser importance.

The principal threats against which any Security Model used within this architecture SHOULD provide protection are:

Modification of Information
: The modification threat is the danger that some unauthorized entity may alter in-transit SNMP messages generated on behalf of an authorized principal in such a way as to effect unauthorized management operations, including falsifying the value of an object.

Masquerade
The masquerade threat is the danger that management operations not authorized for some principal may be attempted by assuming the identity of another principal that has the appropriate authorizations.

Secondary threats against which any Security Model used within this architecture SHOULD provide protection are:

Message Stream Modification
The SNMP protocol is typically based upon a connectionless transport service which may operate over any subnetwork service. The re-ordering, delay or replay of messages can and does occur through the natural operation of many such subnetwork services. The message stream modification threat is the danger that messages may be maliciously re-ordered, delayed or replayed to an extent which is greater than can occur through the natural operation of a subnetwork service, in order to effect unauthorized management operations.

Disclosure
The disclosure threat is the danger of eavesdropping on the exchanges between SNMP engines. Protecting against this threat may be required as a matter of local policy.

There are at least two threats against which a Security Model within this architecture need not protect, since they are deemed to be of lesser importance in this context:

Denial of Service
A Security Model need not attempt to address the broad range of attacks by which service on behalf of authorized users is denied. Indeed, such denial-of-service attacks are in many cases indistinguishable from the type of network failures with which any viable management protocol must cope as a matter of course.

Traffic Analysis
A Security Model need not attempt to address traffic analysis attacks. Many traffic patterns are predictable - entities may be managed on a regular basis by a relatively small number of management stations - and therefore there is no significant advantage afforded by protecting against traffic analysis.

1.5. Design Decisions

Various design decisions were made in support of the goals of the architecture and the security requirements:

- Architecture
  An architecture should be defined which identifies the conceptual boundaries between the documents. Subsystems should be defined which describe the abstract services provided by specific portions of an SNMP framework. Abstract service interfaces, as described by service primitives, define the abstract boundaries between documents, and the abstract services that are provided by the conceptual subsystems of an SNMP framework.

- Self-contained Documents
  Elements of procedure plus the MIB objects which are needed for processing for a specific portion of an SNMP framework should be defined in the same document, and as much as possible, should not be referenced in other documents. This allows pieces to be designed and documented as independent and self-contained parts, which is consistent with the general SNMP MIB module approach. As portions of SNMP change over time, the documents describing other portions of SNMP are not directly impacted. This modularity allows, for example, Security Models, authentication and privacy mechanisms, and message formats to be upgraded and supplemented as the need arises. The self-contained documents can move along the standards track on different time-lines.

  This modularity of specification is not meant to be interpreted as imposing any specific requirements on implementation.

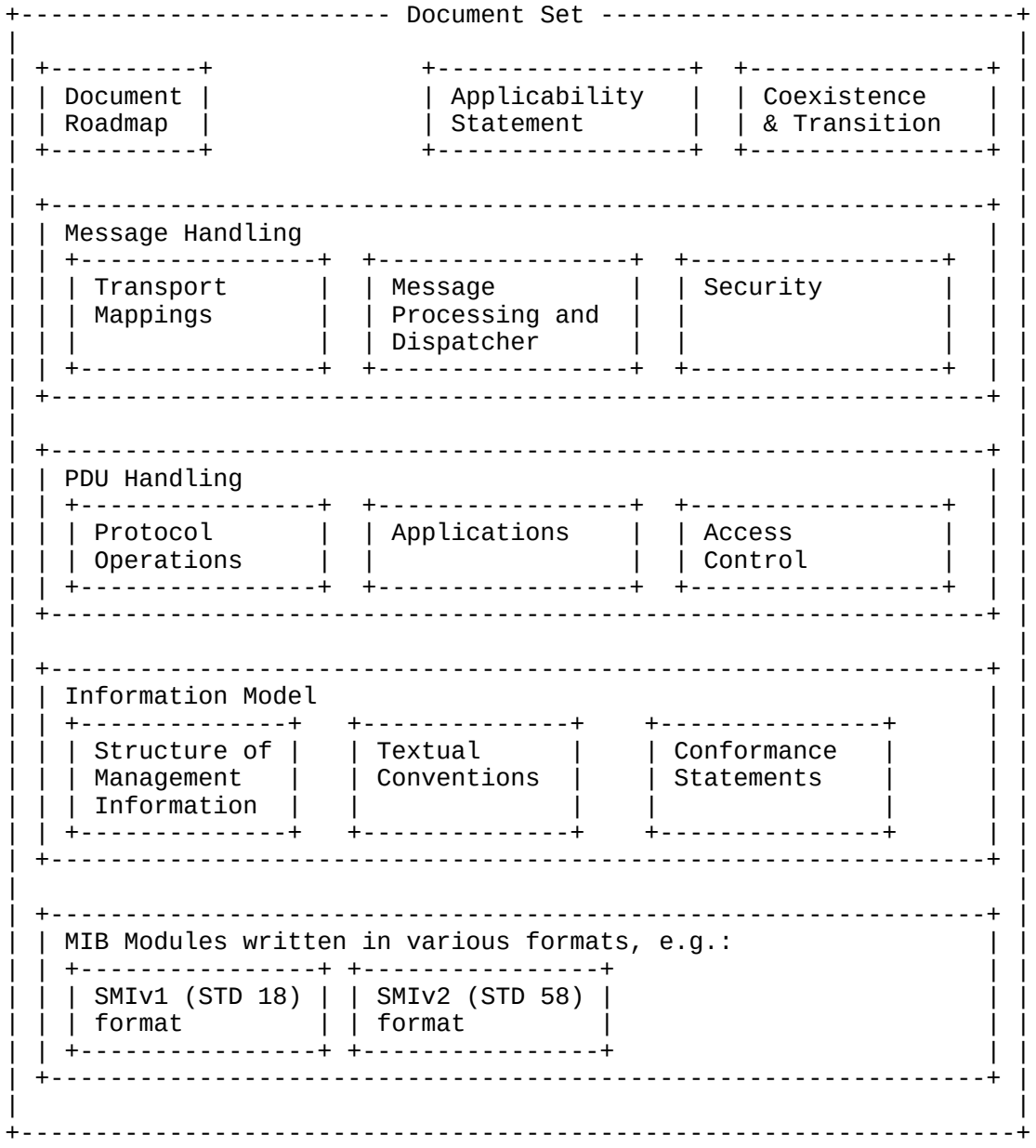
- Threats
  The Security Models in the Security Subsystem SHOULD protect against the principal and secondary threats: modification of information, masquerade, message stream modification and disclosure. They do not need to protect against denial of service and traffic analysis.

- Remote Configuration
  The Security and Access Control Subsystems add a whole new set of SNMP configuration parameters. The Security Subsystem also requires frequent changes of secrets at the various SNMP entities. To make this deployable in a large operational environment, these SNMP parameters must be remotely configurable.

- Controlled Complexity
  It is recognized that producers of simple managed devices want to keep the resources used by SNMP to a minimum. At the same time, there is a need for more complex configurations which can spend more resources for SNMP and thus provide more functionality. The design tries to keep the competing requirements of these two environments in balance and allows the more complex environments to logically extend the simple environment.

2. Documentation Overview

The following figure shows the set of documents that fit within the SNMP Architecture.



Each of these documents may be replaced or supplemented. This Architecture document specifically describes how new documents fit into the set of documents in the area of Message and PDU handling.

2.1. Document Roadmap

One or more documents may be written to describe how sets of documents taken together form specific Frameworks. The configuration of document sets might change over time, so the "road map" should be maintained in a document separate from the standards documents themselves.

An example of such a roadmap is "Introduction and Applicability Statements for the Internet-Standard Management Framework" [RFC3410].

2.2. Applicability Statement

SNMP is used in networks that vary widely in size and complexity, by organizations that vary widely in their requirements of management. Some models will be designed to address specific problems of management, such as message security.

One or more documents may be written to describe the environments to which certain versions of SNMP or models within SNMP would be appropriately applied, and those to which a given model might be inappropriately applied.

2.3. Coexistence and Transition

The purpose of an evolutionary architecture is to permit new models to replace or supplement existing models. The interactions between models could result in incompatibilities, security "holes", and other undesirable effects.

The purpose of Coexistence documents is to detail recognized anomalies and to describe required and recommended behaviors for resolving the interactions between models within the architecture.

Coexistence documents may be prepared separately from model definition documents, to describe and resolve interaction anomalies between a model definition and one or more other model definitions.

Additionally, recommendations for transitions between models may also be described, either in a coexistence document or in a separate document.

One such coexistence document is [RFC2576], "Coexistence between Version 1, Version 2, and Version 3 of the Internet-Standard Network Management Framework".

2.4. Transport Mappings

SNMP messages are sent over various transports. It is the purpose of Transport Mapping documents to define how the mapping between SNMP and the transport is done.

2.5. Message Processing

A Message Processing Model document defines a message format, which is typically identified by a version field in an SNMP message header. The document may also define a MIB module for use in message processing and for instrumentation of version-specific interactions.

An SNMP engine includes one or more Message Processing Models, and thus may support sending and receiving multiple versions of SNMP messages.

2.6. Security

Some environments require secure protocol interactions. Security is normally applied at two different stages:

- in the transmission/receipt of messages, and
- in the processing of the contents of messages.

For purposes of this document, "security" refers to message-level security; "access control" refers to the security applied to protocol operations.

Authentication, encryption, and timeliness checking are common functions of message level security.

A security document describes a Security Model, the threats against which the model protects, the goals of the Security Model, the protocols which it uses to meet those goals, and it may define a MIB module to describe the data used during processing, and to allow the remote configuration of message-level security parameters, such as keys.

An SNMP engine may support multiple Security Models concurrently.

2.7. Access Control

During processing, it may be required to control access to managed objects for operations.

An Access Control Model defines mechanisms to determine whether access to a managed object should be allowed. An Access Control Model may define a MIB module used during processing and to allow the remote configuration of access control policies.

2.8. Protocol Operations

SNMP messages encapsulate an SNMP Protocol Data Unit (PDU). SNMP PDUs define the operations performed by the receiving SNMP engine. It is the purpose of a Protocol Operations document to define the operations of the protocol with respect to the processing of the PDUs. Every PDU belongs to one or more of the PDU classes defined below:

1) Read Class:

   The Read Class contains protocol operations that retrieve management information. For example, [RFC3416] defines the following protocol operations for the Read Class: GetRequest-PDU, GetNextRequest-PDU, and GetBulkRequest-PDU.

2) Write Class:

   The Write Class contains protocol operations which attempt to modify management information. For example, [RFC3416] defines the following protocol operation for the Write Class: SetRequest-PDU.

3) Response Class:

   The Response Class contains protocol operations which are sent in response to a previous request. For example, [RFC3416] defines the following for the Response Class: Response-PDU, Report-PDU.

4) Notification Class:

   The Notification Class contains protocol operations which send a notification to a notification receiver application. For example, [RFC3416] defines the following operations for the Notification Class: Trapv2-PDU, InformRequest-PDU.

5) Internal Class:

The Internal Class contains protocol operations which are exchanged internally between SNMP engines. For example, [RFC3416] defines the following operation for the Internal Class: Report-PDU.

The preceding five classifications are based on the functional properties of a PDU. It is also useful to classify PDUs based on whether a response is expected:

6) Confirmed Class:

The Confirmed Class contains all protocol operations which cause the receiving SNMP engine to send back a response. For example, [RFC3416] defines the following operations for the Confirmed Class: GetRequest-PDU, GetNextRequest-PDU, GetBulkRequest-PDU, SetRequest-PDU, and InformRequest-PDU.

7) Unconfirmed Class:

The Unconfirmed Class contains all protocol operations which are not acknowledged. For example, [RFC3416] defines the following operations for the Unconfirmed Class: Report-PDU, Trapv2-PDU, and GetResponse-PDU.

An application document defines which Protocol Operations are supported by the application.

2.9. Applications

An SNMP entity normally includes a number of applications. Applications use the services of an SNMP engine to accomplish specific tasks. They coordinate the processing of management information operations, and may use SNMP messages to communicate with other SNMP entities.

An applications document describes the purpose of an application, the services required of the associated SNMP engine, and the protocol operations and informational model that the application uses to perform management operations.

An application document defines which set of documents are used to specifically define the structure of management information, textual conventions, conformance requirements, and operations supported by the application.

2.10. Structure of Management Information

Management information is viewed as a collection of managed objects, residing in a virtual information store, termed the Management Information Base (MIB). Collections of related objects are defined in MIB modules.

It is the purpose of a Structure of Management Information document to establish the notation for defining objects, modules, and other elements of managed information.

2.11. Textual Conventions

When designing a MIB module, it is often useful to define new types similar to those defined in the SMI, but with more precise semantics, or which have special semantics associated with them. These newly defined types are termed textual conventions, and may be defined in separate documents, or within a MIB module.

2.12. Conformance Statements

It may be useful to define the acceptable lower-bounds of implementation, along with the actual level of implementation achieved. It is the purpose of the Conformance Statements document to define the notation used for these purposes.

2.13. Management Information Base Modules

MIB documents describe collections of managed objects which instrument some aspect of a managed node.

2.13.1. SNMP Instrumentation MIBs

An SNMP MIB document may define a collection of managed objects which instrument the SNMP protocol itself. In addition, MIB modules may be defined within the documents which describe portions of the SNMP architecture, such as the documents for Message processing Models, Security Models, etc. for the purpose of instrumenting those Models, and for the purpose of allowing their remote configuration.

2.14. SNMP Framework Documents

This architecture is designed to allow an orderly evolution of portions of SNMP Frameworks.

Throughout the rest of this document, the term "subsystem" refers to an abstract and incomplete specification of a portion of a Framework, that is further refined by a model specification.

A "model" describes a specific design of a subsystem, defining additional constraints and rules for conformance to the model. A model is sufficiently detailed to make it possible to implement the specification.

An "implementation" is an instantiation of a subsystem, conforming to one or more specific models.

SNMP version 1 (SNMPv1), is the original Internet-Standard Network Management Framework, as described in RFCs 1155, 1157, and 1212.

SNMP version 2 (SNMPv2), is the SNMPv2 Framework as derived from the SNMPv1 Framework. It is described in STD 58, RFCs 2578, 2579, 2580, and STD 62, RFCs 3416, 3417, and 3418. SNMPv2 has no message definition.

The Community-based SNMP version 2 (SNMPv2c), is an experimental SNMP Framework which supplements the SNMPv2 Framework, as described in [RFC1901]. It adds the SNMPv2c message format, which is similar to the SNMPv1 message format.

SNMP version 3 (SNMPv3), is an extensible SNMP Framework which supplements the SNMPv2 Framework, by supporting the following:

- a new SNMP message format,
- Security for Messages,
- Access Control, and
- Remote configuration of SNMP parameters.

Other SNMP Frameworks, i.e., other configurations of implemented subsystems, are expected to also be consistent with this architecture.

3. Elements of the Architecture

This section describes the various elements of the architecture and how they are named. There are three kinds of naming:

1) the naming of entities,

2) the naming of identities, and

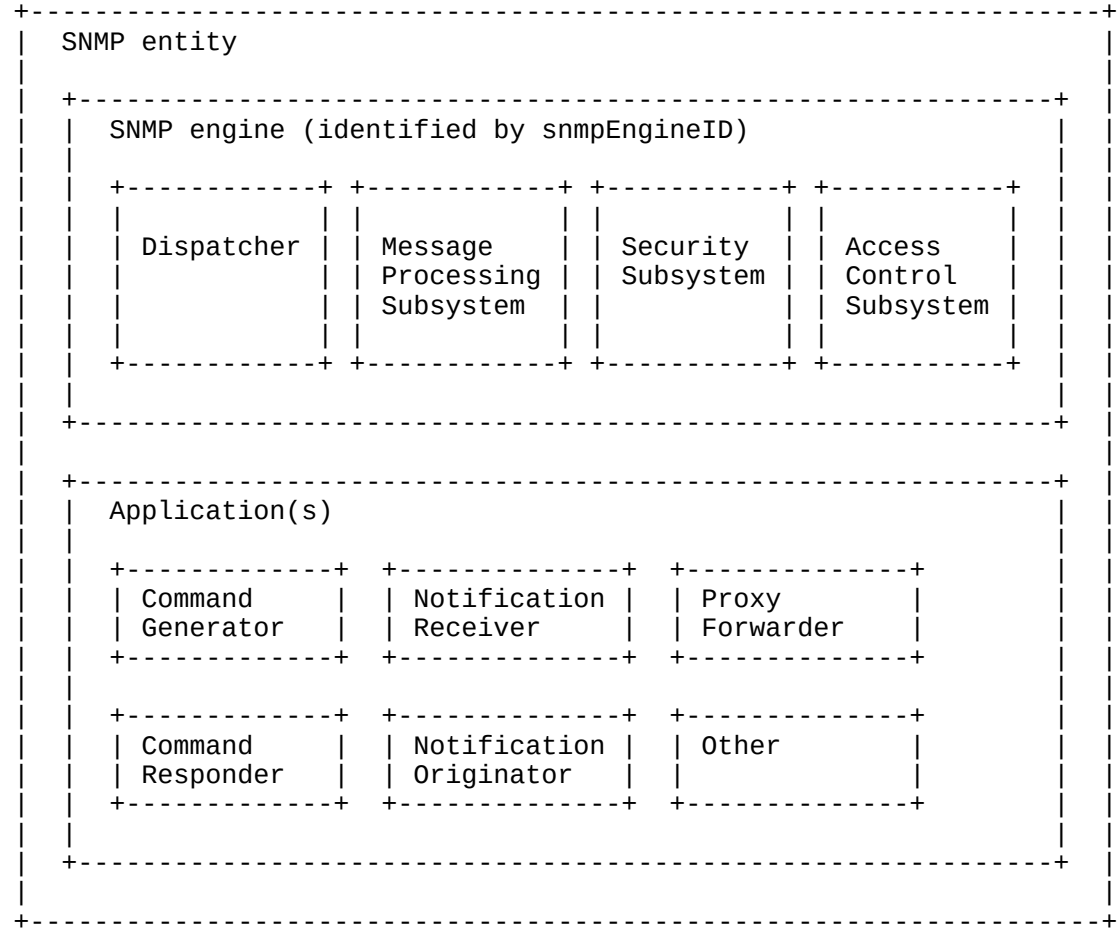
3) the naming of management information.

This architecture also defines some names for other constructs that are used in the documentation.

3.1. The Naming of Entities

An SNMP entity is an implementation of this architecture. Each such SNMP entity consists of an SNMP engine and one or more associated applications.

The following figure shows details about an SNMP entity and the components within it.



3.1.1. SNMP engine

An SNMP engine provides services for sending and receiving messages, authenticating and encrypting messages, and controlling access to managed objects. There is a one-to-one association between an SNMP engine and the SNMP entity which contains it.

The engine contains:

1) a Dispatcher,

2) a Message Processing Subsystem,

3) a Security Subsystem, and

4) an Access Control Subsystem.

3.1.1.1. snmpEngineID

Within an administrative domain, an snmpEngineID is the unique and unambiguous identifier of an SNMP engine. Since there is a one-to-one association between SNMP engines and SNMP entities, it also uniquely and unambiguously identifies the SNMP entity within that administrative domain. Note that it is possible for SNMP entities in different administrative domains to have the same value for snmpEngineID. Federation of administrative domains may necessitate assignment of new values.

3.1.1.2. Dispatcher

There is only one Dispatcher in an SNMP engine. It allows for concurrent support of multiple versions of SNMP messages in the SNMP engine. It does so by:

- sending and receiving SNMP messages to/from the network,

- determining the version of an SNMP message and interacting with the corresponding Message Processing Model,

- providing an abstract interface to SNMP applications for delivery of a PDU to an application.

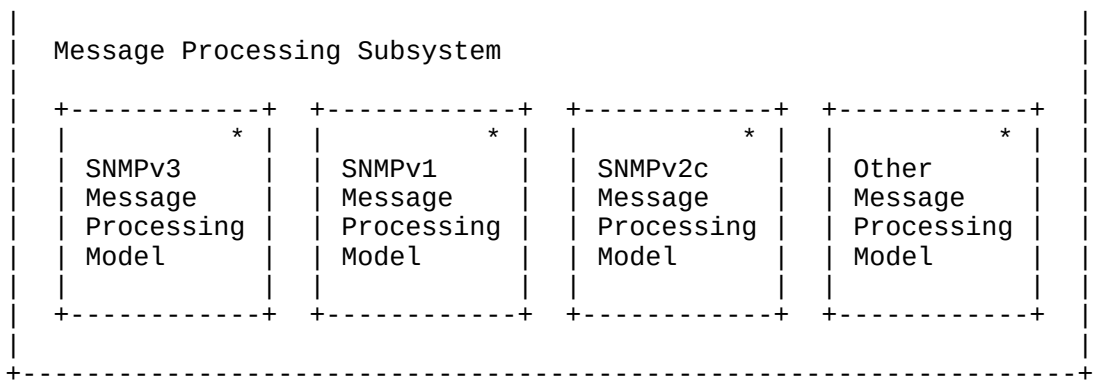
- providing an abstract interface for SNMP applications that allows them to send a PDU to a remote SNMP entity.

3.1.1.3. Message Processing Subsystem

The Message Processing Subsystem is responsible for preparing messages for sending, and extracting data from received messages.

The Message Processing Subsystem potentially contains multiple Message Processing Models as shown in the next figure.

* One or more Message Processing Models may be present.



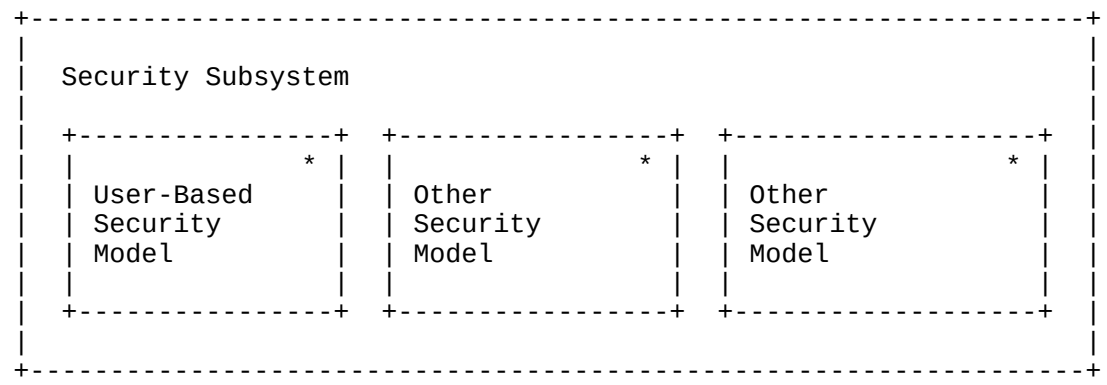
3.1.1.3.1. Message Processing Model

Each Message Processing Model defines the format of a particular version of an SNMP message and coordinates the preparation and extraction of each such version-specific message format.

3.1.1.4. Security Subsystem

The Security Subsystem provides security services such as the authentication and privacy of messages and potentially contains multiple Security Models as shown in the following figure

* One or more Security Models may be present.



3.1.1.4.1. Security Model

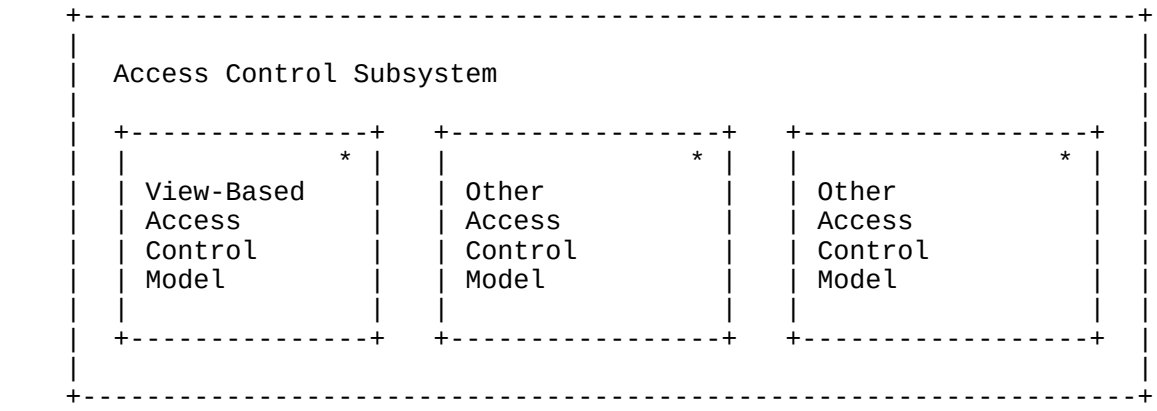
A Security Model specifies the threats against which it protects, the goals of its services, and the security protocols used to provide security services such as authentication and privacy.

3.1.1.4.2. Security Protocol

A Security Protocol specifies the mechanisms, procedures, and MIB objects used to provide a security service such as authentication or privacy.

3.1.2. Access Control Subsystem

The Access Control Subsystem provides authorization services by means of one or more (*) Access Control Models.



3.1.2.1. Access Control Model

An Access Control Model defines a particular access decision function in order to support decisions regarding access rights.

3.1.3. Applications

There are several types of applications, including:

- command generators, which monitor and manipulate management data,
- command responders, which provide access to management data,
- notification originators, which initiate asynchronous messages,
- notification receivers, which process asynchronous messages,

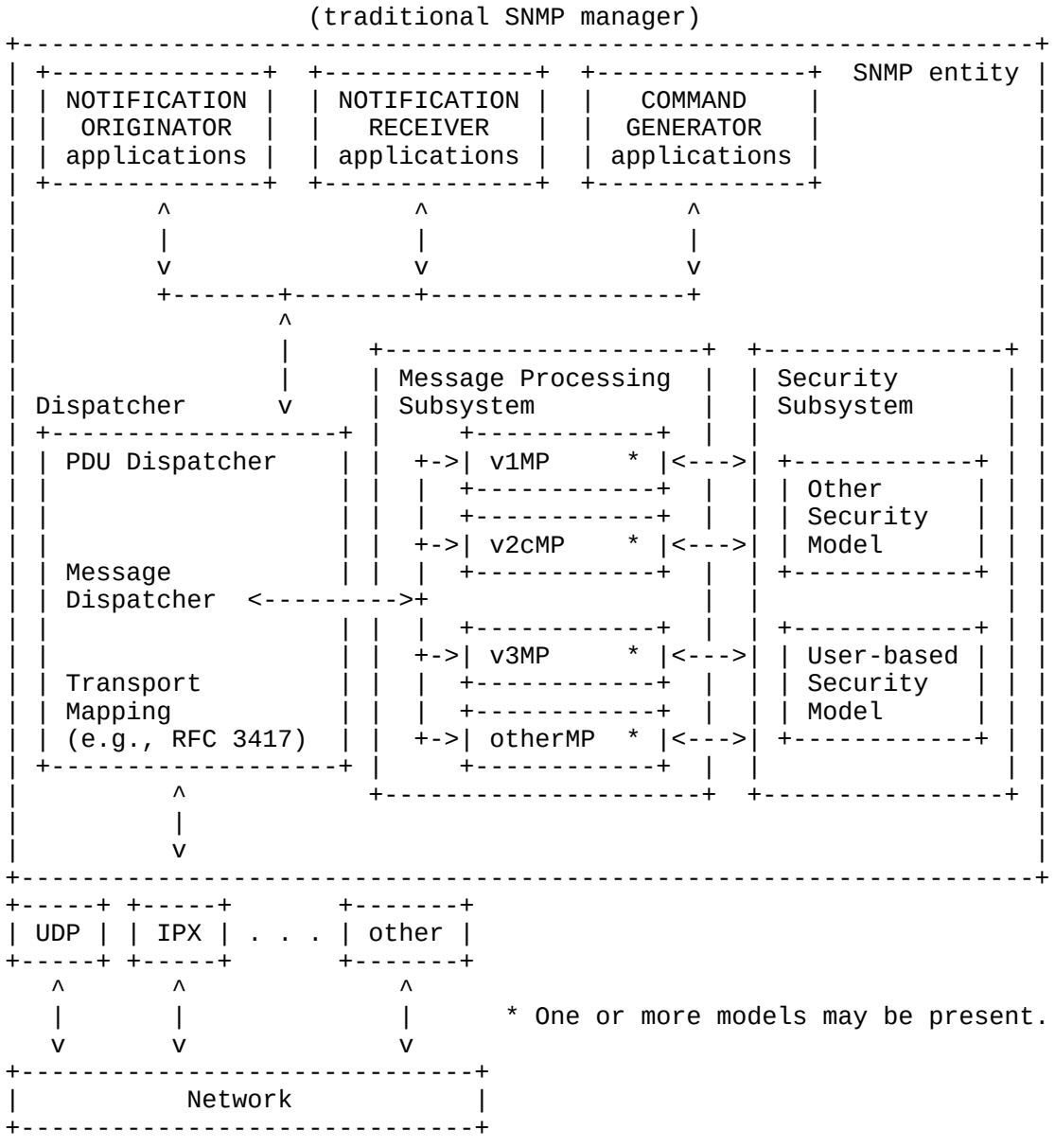
and

- proxy forwarders, which forward messages between entities.

These applications make use of the services provided by the SNMP engine.

3.1.3.1. SNMP Manager

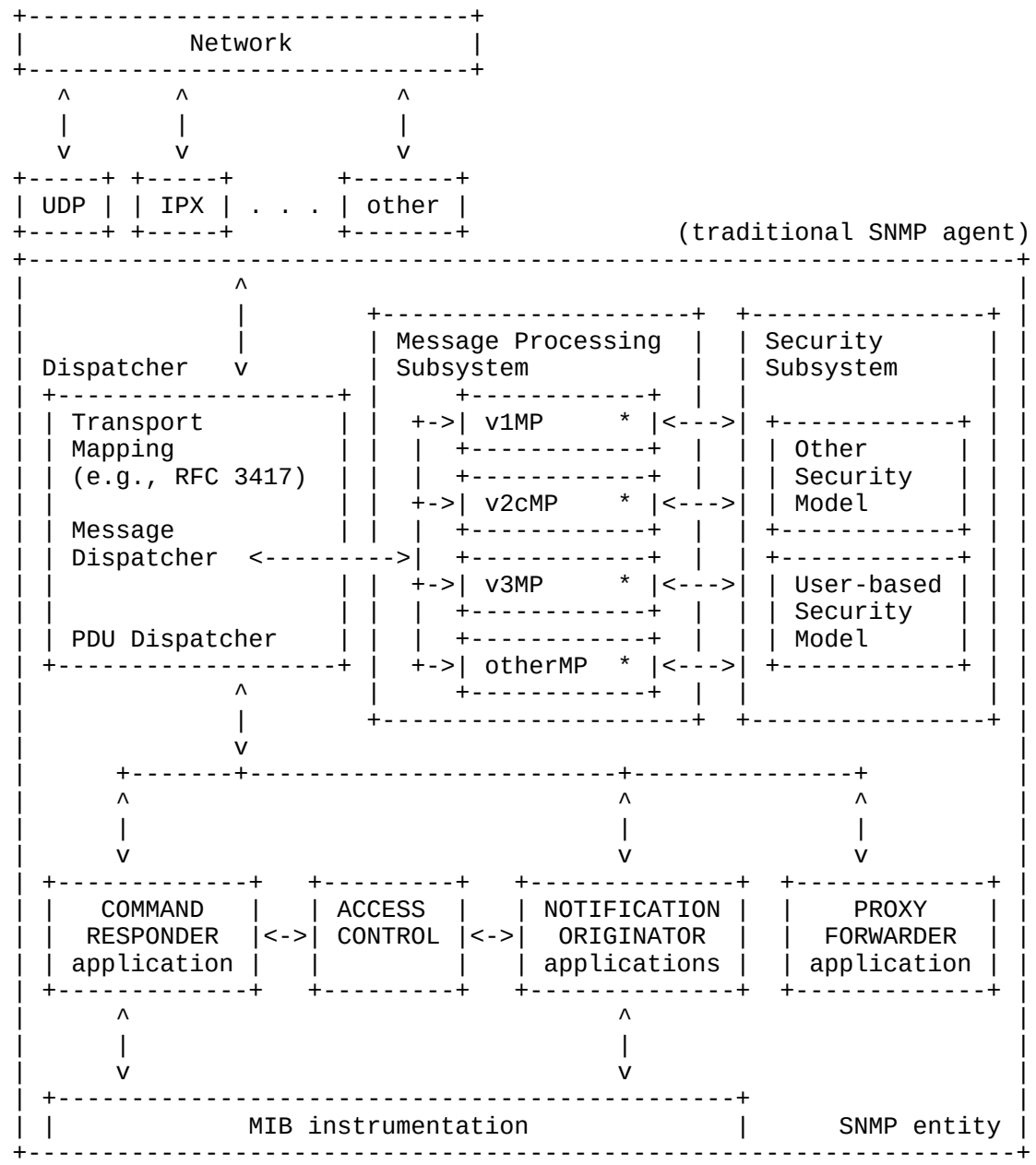
An SNMP entity containing one or more command generator and/or notification receiver applications (along with their associated SNMP engine) has traditionally been called an SNMP manager.



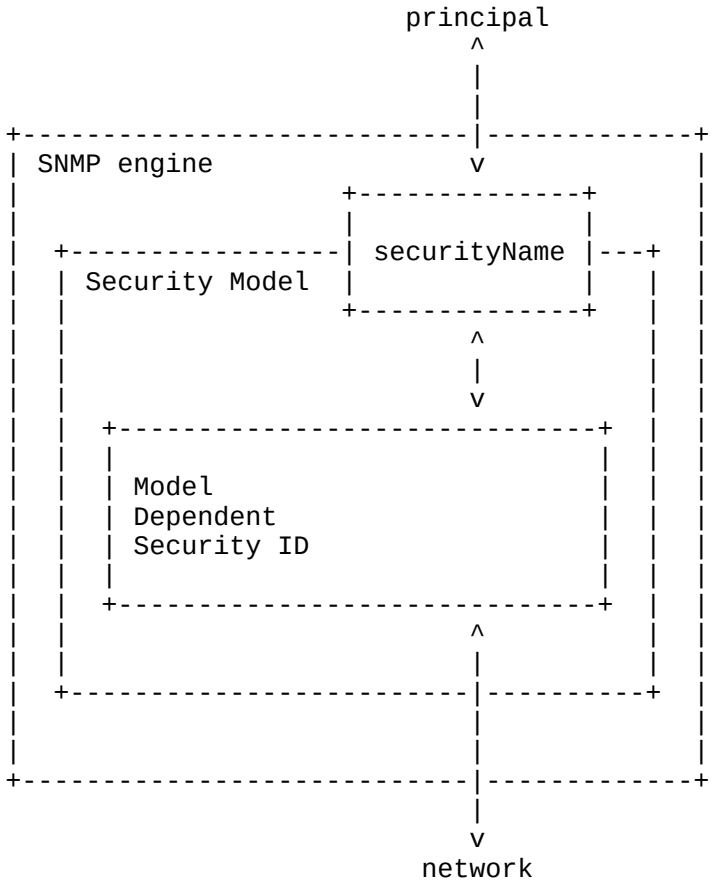
3.1.3.2. SNMP Agent

An SNMP entity containing one or more command responder and/or notification originator applications (along with their associated SNMP engine) has traditionally been called an SNMP agent.

* One or more models may be present.



3.2. The Naming of Identities



3.2.1. Principal

A principal is the "who" on whose behalf services are provided or processing takes place.

A principal can be, among other things, an individual acting in a particular role; a set of individuals, with each acting in a particular role; an application or a set of applications; and combinations thereof.

3.2.2. securityName

A securityName is a human readable string representing a principal. It has a model-independent format, and can be used outside a particular Security Model.

3.2.3. Model-dependent security ID

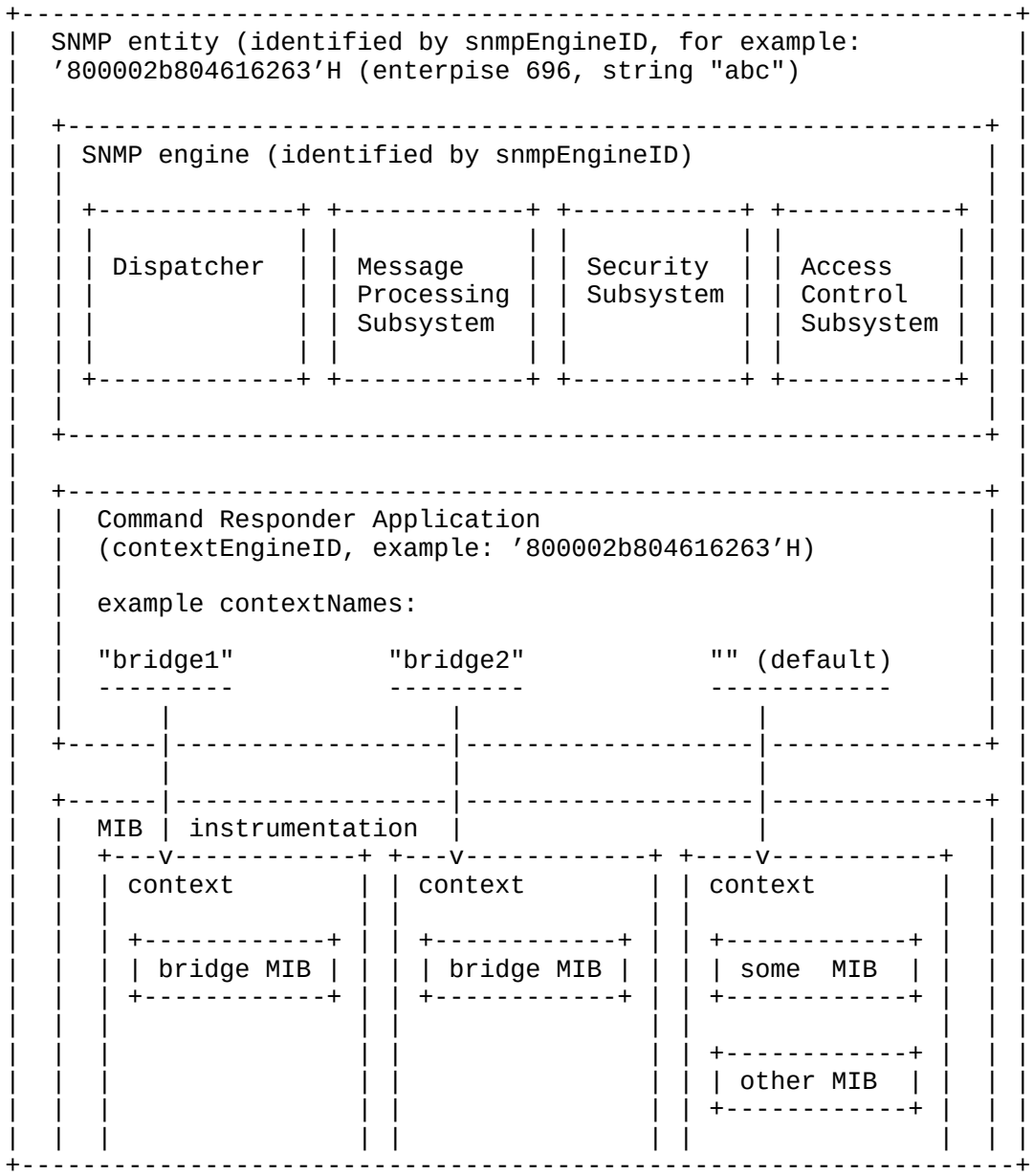
A model-dependent security ID is the model-specific representation of a securityName within a particular Security Model.

Model-dependent security IDs may or may not be human readable, and have a model-dependent syntax. Examples include community names, and user names.

The transformation of model-dependent security IDs into securityNames and vice versa is the responsibility of the relevant Security Model.

3.3. The Naming of Management Information

Management information resides at an SNMP entity where a Command Responder Application has local access to potentially multiple contexts. This application uses a contextEngineID equal to the snmpEngineID of its associated SNMP engine.


SNMP entity (identified by snmpEngineID, for example:
'800002b804616263'H (enterpise 696, string "abc")
SNMP engine (identified by snmpEngineID)
Dispatcher
Message Processing Subsystem
Security Subsystem
Access Control Subsystem
Command Responder Application
(contextEngineID, example: '800002b804616263'H)
example contextNames:
"bridge1"
"bridge2"
"" (default)
MIB | instrumentation
context
bridge MIB
context
bridge MIB
context
some MIB
other MIB

3.3.1. An SNMP Context

An SNMP context, or just "context" for short, is a collection of management information accessible by an SNMP entity. An item of management information may exist in more than one context. An SNMP entity potentially has access to many contexts.

Typically, there are many instances of each managed object type within a management domain. For simplicity, the method for identifying instances specified by the MIB module does not allow each instance to be distinguished amongst the set of all instances within a management domain; rather, it allows each instance to be identified only within some scope or "context", where there are multiple such contexts within the management domain. Often, a context is a physical device, or perhaps, a logical device, although a context can also encompass multiple devices, or a subset of a single device, or even a subset of multiple devices, but a context is always defined as a subset of a single SNMP entity. Thus, in order to identify an individual item of management information within the management domain, its contextName and contextEngineID must be identified in addition to its object type and its instance.

For example, the managed object type ifDescr [RFC2863], is defined as the description of a network interface. To identify the description of device-X's first network interface, four pieces of information are needed: the snmpEngineID of the SNMP entity which provides access to the management information at device-X, the contextName (device-X), the managed object type (ifDescr), and the instance ("1").

Each context has (at least) one unique identification within the management domain. The same item of management information can exist in multiple contexts. An item of management information may have multiple unique identifications. This occurs when an item of management information exists in multiple contexts, and this also occurs when a context has multiple unique identifications.

The combination of a contextEngineID and a contextName unambiguously identifies a context within an administrative domain; note that there may be multiple unique combinations of contextEngineID and contextName that unambiguously identify the same context.

3.3.2. contextEngineID

Within an administrative domain, a contextEngineID uniquely identifies an SNMP entity that may realize an instance of a context with a particular contextName.

3.3.3. contextName

A contextName is used to name a context. Each contextName MUST be unique within an SNMP entity.

3.3.4. scopedPDU

A scopedPDU is a block of data containing a contextEngineID, a contextName, and a PDU.

The PDU is an SNMP Protocol Data Unit containing information named in the context which is unambiguously identified within an administrative domain by the combination of the contextEngineID and the contextName. See, for example, RFC 3416 for more information about SNMP PDUs.

3.4. Other Constructs

3.4.1. maxSizeResponseScopedPDU

The maxSizeResponseScopedPDU is the maximum size of a scopedPDU that a PDU's sender would be willing to accept. Note that the size of a scopedPDU does not include the size of the SNMP message header.

3.4.2. Local Configuration Datastore

The subsystems, models, and applications within an SNMP entity may need to retain their own sets of configuration information.

Portions of the configuration information may be accessible as managed objects.

The collection of these sets of information is referred to as an entity's Local Configuration Datastore (LCD).

3.4.3. securityLevel

This architecture recognizes three levels of security:

- without authentication and without privacy (noAuthNoPriv)
- with authentication but without privacy (authNoPriv)
- with authentication and with privacy (authPriv)

These three values are ordered such that noAuthNoPriv is less than authNoPriv and authNoPriv is less than authPriv.

Every message has an associated securityLevel. All Subsystems (Message Processing, Security, Access Control) and applications are REQUIRED to either supply a value of securityLevel or to abide by the supplied value of securityLevel while processing the message and its contents.

4. Abstract Service Interfaces

Abstract service interfaces have been defined to describe the conceptual interfaces between the various subsystems within an SNMP entity. The abstract service interfaces are intended to help clarify the externally observable behavior of SNMP entities, and are not intended to constrain the structure or organization of implementations in any way. Most specifically, they should not be interpreted as APIs or as requirements statements for APIs.

These abstract service interfaces are defined by a set of primitives that define the services provided and the abstract data elements that are to be passed when the services are invoked. This section lists the primitives that have been defined for the various subsystems.

4.1. Dispatcher Primitives

The Dispatcher typically provides services to the SNMP applications via its PDU Dispatcher. This section describes the primitives provided by the PDU Dispatcher.

4.1.1. Generate Outgoing Request or Notification

The PDU Dispatcher provides the following primitive for an application to send an SNMP Request or Notification to another SNMP entity:

```
statusInformation =              -- sendPduHandle if success
                                 -- errorIndication if failure
  sendPdu(
  IN   transportDomain           -- transport domain to be used
  IN   transportAddress          -- transport address to be used
  IN   messageProcessingModel    -- typically, SNMP version
  IN   securityModel             -- Security Model to use
  IN   securityName              -- on behalf of this principal
  IN   securityLevel             -- Level of Security requested
  IN   contextEngineID           -- data from/at this entity
  IN   contextName               -- data from/in this context
  IN   pduVersion                -- the version of the PDU
  IN   PDU                       -- SNMP Protocol Data Unit
  IN   expectResponse            -- TRUE or FALSE
       )
```

4.1.2. Process Incoming Request or Notification PDU

The PDU Dispatcher provides the following primitive to pass an incoming SNMP PDU to an application:

```
processPdu(                      -- process Request/Notification PDU
  IN   messageProcessingModel    -- typically, SNMP version
  IN   securityModel             -- Security Model in use
  IN   securityName              -- on behalf of this principal
  IN   securityLevel             -- Level of Security
  IN   contextEngineID           -- data from/at this SNMP entity
  IN   contextName               -- data from/in this context
  IN   pduVersion                -- the version of the PDU
  IN   PDU                       -- SNMP Protocol Data Unit
  IN   maxSizeResponseScopedPDU  -- maximum size of the Response PDU
  IN   stateReference            -- reference to state information
       )                         -- needed when sending a response
```

4.1.3. Generate Outgoing Response

The PDU Dispatcher provides the following primitive for an application to return an SNMP Response PDU to the PDU Dispatcher:

```
result =                          -- SUCCESS or FAILURE
returnResponsePdu(
  IN   messageProcessingModel     -- typically, SNMP version
  IN   securityModel              -- Security Model in use
  IN   securityName               -- on behalf of this principal
  IN   securityLevel              -- same as on incoming request
  IN   contextEngineID            -- data from/at this SNMP entity
  IN   contextName                -- data from/in this context
  IN   pduVersion                 -- the version of the PDU
  IN   PDU                        -- SNMP Protocol Data Unit
  IN   maxSizeResponseScopedPDU   -- maximum size sender can accept
  IN   stateReference             -- reference to state information
                                  -- as presented with the request
  IN   statusInformation          -- success or errorIndication
       )                          -- error counter OID/value if error
```

4.1.4. Process Incoming Response PDU

The PDU Dispatcher provides the following primitive to pass an incoming SNMP Response PDU to an application:

```
processResponsePdu(               -- process Response PDU
  IN   messageProcessingModel     -- typically, SNMP version
  IN   securityModel              -- Security Model in use
  IN   securityName               -- on behalf of this principal
  IN   securityLevel              -- Level of Security
  IN   contextEngineID            -- data from/at this SNMP entity
  IN   contextName                -- data from/in this context
  IN   pduVersion                 -- the version of the PDU
  IN   PDU                        -- SNMP Protocol Data Unit
  IN   statusInformation          -- success or errorIndication
  IN   sendPduHandle              -- handle from sendPdu
       )
```

4.1.5. Registering Responsibility for Handling SNMP PDUs

Applications can register/unregister responsibility for a specific contextEngineID, for specific pduTypes, with the PDU Dispatcher according to the following primitives. The list of particular pduTypes that an application can register for is determined by the Message Processing Model(s) supported by the SNMP entity that contains the PDU Dispatcher.

```
statusInformation =            -- success or errorIndication
  registerContextEngineID(
  IN   contextEngineID         -- take responsibility for this one
  IN   pduType                 -- the pduType(s) to be registered
       )

unregisterContextEngineID(
  IN   contextEngineID         -- give up responsibility for this one
  IN   pduType                 -- the pduType(s) to be unregistered
       )
```

Note that realizations of the registerContextEngineID and unregisterContextEngineID abstract service interfaces may provide implementation-specific ways for applications to register/deregister responsibility for all possible values of the contextEngineID or pduType parameters.

4.2. Message Processing Subsystem Primitives

The Dispatcher interacts with a Message Processing Model to process a specific version of an SNMP Message. This section describes the primitives provided by the Message Processing Subsystem.

4.2.1. Prepare Outgoing SNMP Request or Notification Message

The Message Processing Subsystem provides this service primitive for preparing an outgoing SNMP Request or Notification Message:

```
statusInformation =              -- success or errorIndication
  prepareOutgoingMessage(
  IN   transportDomain           -- transport domain to be used
  IN   transportAddress          -- transport address to be used
  IN   messageProcessingModel    -- typically, SNMP version
  IN   securityModel             -- Security Model to use
  IN   securityName              -- on behalf of this principal
  IN   securityLevel             -- Level of Security requested
  IN   contextEngineID           -- data from/at this entity
  IN   contextName               -- data from/in this context
  IN   pduVersion                -- the version of the PDU
  IN   PDU                       -- SNMP Protocol Data Unit
  IN   expectResponse            -- TRUE or FALSE
  IN   sendPduHandle             -- the handle for matching
                                 -- incoming responses
  OUT  destTransportDomain       -- destination transport domain
  OUT  destTransportAddress      -- destination transport address
  OUT  outgoingMessage           -- the message to send
  OUT  outgoingMessageLength     -- its length
       )
```

4.2.2. Prepare an Outgoing SNMP Response Message

The Message Processing Subsystem provides this service primitive for preparing an outgoing SNMP Response Message:

```
result =                          -- SUCCESS or FAILURE
  prepareResponseMessage(
  IN   messageProcessingModel     -- typically, SNMP version
  IN   securityModel              -- same as on incoming request
  IN   securityName               -- same as on incoming request
  IN   securityLevel              -- same as on incoming request
  IN   contextEngineID            -- data from/at this SNMP entity
  IN   contextName                -- data from/in this context
  IN   pduVersion                 -- the version of the PDU
  IN   PDU                        -- SNMP Protocol Data Unit
  IN   maxSizeResponseScopedPDU   -- maximum size able to accept
  IN   stateReference             -- reference to state information
                                  -- as presented with the request
  IN   statusInformation          -- success or errorIndication
                                  -- error counter OID/value if error
  OUT  destTransportDomain        -- destination transport domain
  OUT  destTransportAddress       -- destination transport address
  OUT  outgoingMessage            -- the message to send
  OUT  outgoingMessageLength      -- its length
       )
```

4.2.3. Prepare Data Elements from an Incoming SNMP Message

The Message Processing Subsystem provides this service primitive for preparing the abstract data elements from an incoming SNMP message:

```
result =                          -- SUCCESS or errorIndication
  prepareDataElements(
  IN   transportDomain            -- origin transport domain
  IN   transportAddress           -- origin transport address
  IN   wholeMsg                   -- as received from the network
  IN   wholeMsgLength             -- as received from the network
  OUT  messageProcessingModel     -- typically, SNMP version
  OUT  securityModel              -- Security Model to use
  OUT  securityName               -- on behalf of this principal
  OUT  securityLevel              -- Level of Security requested
  OUT  contextEngineID            -- data from/at this entity
  OUT  contextName                -- data from/in this context
  OUT  pduVersion                 -- the version of the PDU
  OUT  PDU                        -- SNMP Protocol Data Unit
  OUT  pduType                    -- SNMP PDU type
  OUT  sendPduHandle              -- handle for matched request
  OUT  maxSizeResponseScopedPDU   -- maximum size sender can accept
  OUT  statusInformation          -- success or errorIndication
                                  -- error counter OID/value if error
  OUT  stateReference             -- reference to state information
                                  -- to be used for possible Response
       )
```

4.3. Access Control Subsystem Primitives

Applications are the typical clients of the service(s) of the Access Control Subsystem.

The following primitive is provided by the Access Control Subsystem to check if access is allowed:

```
statusInformation =               -- success or errorIndication
  isAccessAllowed(
  IN   securityModel              -- Security Model in use
  IN   securityName               -- principal who wants to access
  IN   securityLevel              -- Level of Security
  IN   viewType                   -- read, write, or notify view
  IN   contextName                -- context containing variableName
  IN   variableName               -- OID for the managed object
       )
```

4.4. Security Subsystem Primitives

The Message Processing Subsystem is the typical client of the services of the Security Subsystem.

4.4.1. Generate a Request or Notification Message

The Security Subsystem provides the following primitive to generate a Request or Notification message:

```
statusInformation =
  generateRequestMsg(
  IN   messageProcessingModel    -- typically, SNMP version
  IN   globalData                -- message header, admin data
  IN   maxMessageSize            -- of the sending SNMP entity
  IN   securityModel             -- for the outgoing message
  IN   securityEngineID          -- authoritative SNMP entity
  IN   securityName              -- on behalf of this principal
  IN   securityLevel             -- Level of Security requested
  IN   scopedPDU                 -- message (plaintext) payload
  OUT  securityParameters        -- filled in by Security Module
  OUT  wholeMsg                  -- complete generated message
  OUT  wholeMsgLength            -- length of the generated message
       )
```

4.4.2. Process Incoming Message

The Security Subsystem provides the following primitive to process an incoming message:

```
statusInformation =              -- errorIndication or success
                                 -- error counter OID/value if error
  processIncomingMsg(
  IN   messageProcessingModel    -- typically, SNMP version
  IN   maxMessageSize            -- of the sending SNMP entity
  IN   securityParameters        -- for the received message
  IN   securityModel             -- for the received message
  IN   securityLevel             -- Level of Security
  IN   wholeMsg                  -- as received on the wire
  IN   wholeMsgLength            -- length as received on the wire
  OUT  securityEngineID          -- authoritative SNMP entity
  OUT  securityName              -- identification of the principal
  OUT  scopedPDU,                -- message (plaintext) payload
  OUT  maxSizeResponseScopedPDU  -- maximum size sender can handle
  OUT  securityStateReference    -- reference to security state
       )                         -- information, needed for response
```

4.4.3. Generate a Response Message

The Security Subsystem provides the following primitive to generate a Response message:

```
statusInformation =
  generateResponseMsg(
  IN   messageProcessingModel    -- typically, SNMP version
  IN   globalData                -- message header, admin data
  IN   maxMessageSize            -- of the sending SNMP entity
  IN   securityModel             -- for the outgoing message
  IN   securityEngineID          -- authoritative SNMP entity
  IN   securityName              -- on behalf of this principal
  IN   securityLevel             -- for the outgoing message
  IN   scopedPDU                 -- message (plaintext) payload
  IN   securityStateReference    -- reference to security state
                                 -- information from original request
  OUT  securityParameters        -- filled in by Security Module
  OUT  wholeMsg                  -- complete generated message
  OUT  wholeMsgLength            -- length of the generated message
       )
```

4.5. Common Primitives

These primitive(s) are provided by multiple Subsystems.

4.5.1. Release State Reference Information

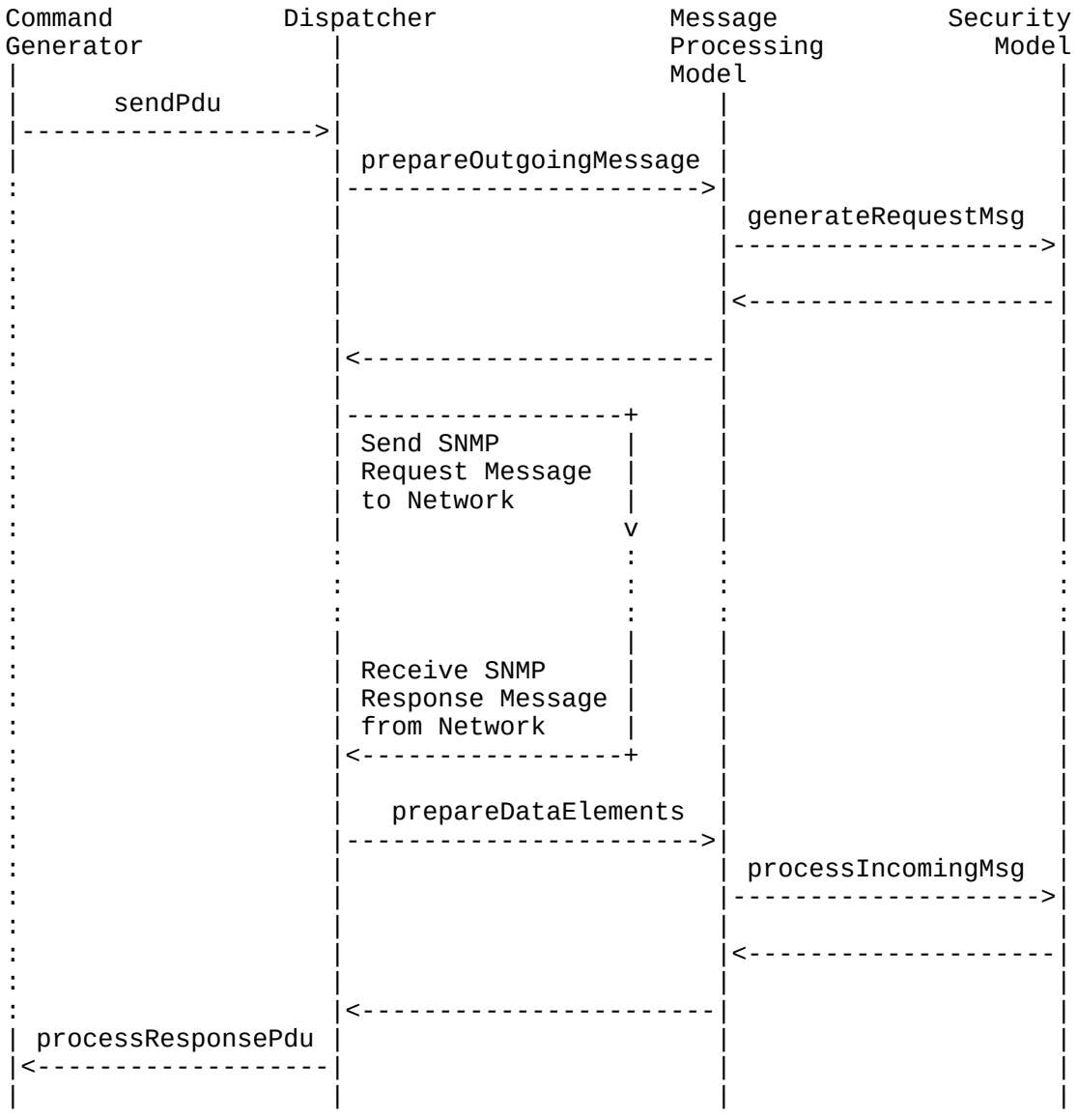
All Subsystems which pass stateReference information also provide a primitive to release the memory that holds the referenced state information:

```
stateRelease(
  IN   stateReference       -- handle of reference to be released
       )
```

4.6. Scenario Diagrams

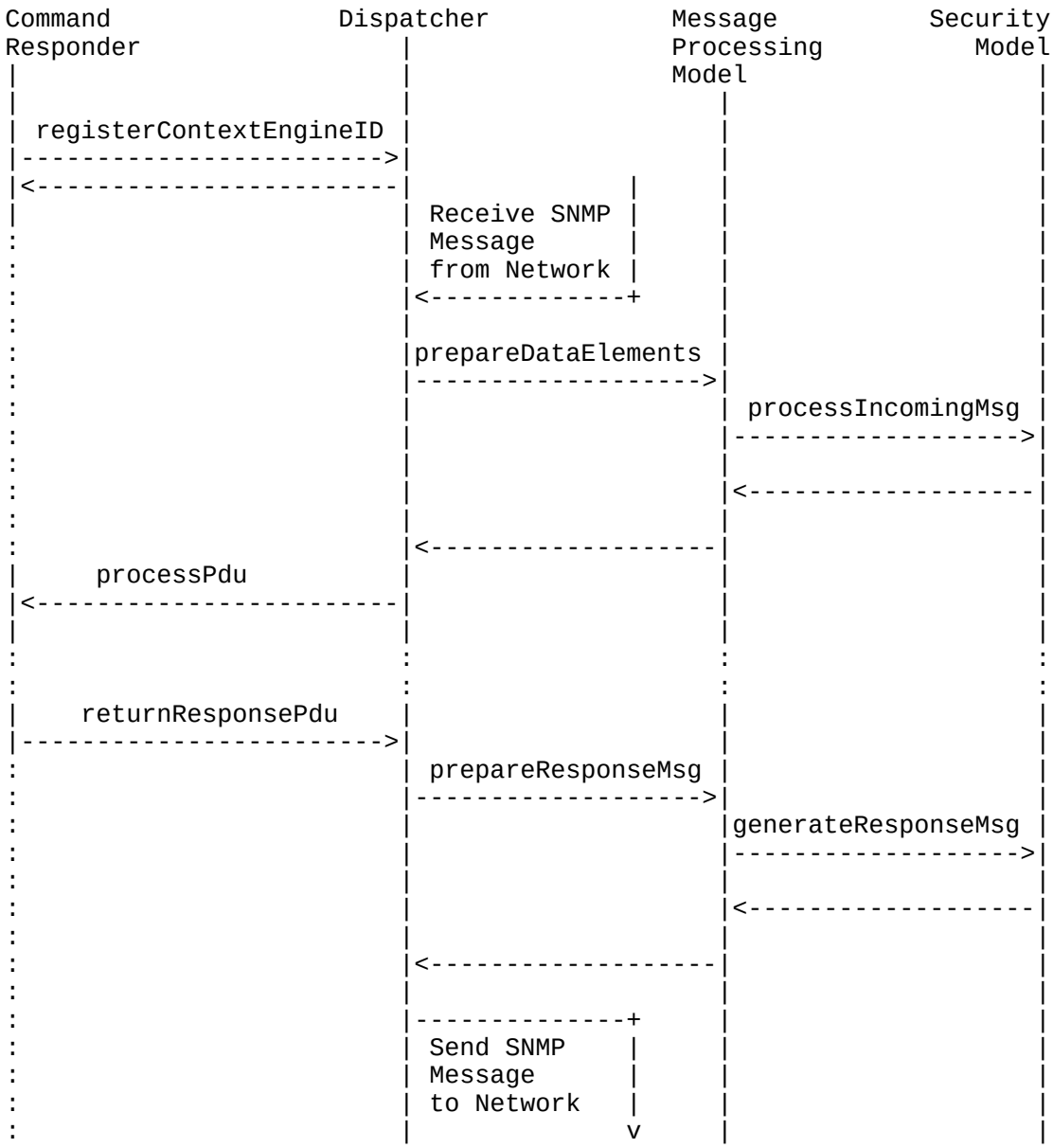
4.6.1. Command Generator or Notification Originator

This diagram shows how a Command Generator or Notification Originator application requests that a PDU be sent, and how the response is returned (asynchronously) to that application.



4.6.2. Scenario Diagram for a Command Responder Application

This diagram shows how a Command Responder or Notification Receiver application registers for handling a pduType, how a PDU is dispatched to the application after an SNMP message is received, and how the Response is (asynchronously) send back to the network.



5. Managed Object Definitions for SNMP Management Frameworks

```
SNMP-FRAMEWORK-MIB DEFINITIONS ::= BEGIN

IMPORTS
    MODULE-IDENTITY, OBJECT-TYPE,
    OBJECT-IDENTITY,
    snmpModules                          FROM SNMPv2-SMI
    TEXTUAL-CONVENTION                   FROM SNMPv2-TC
    MODULE-COMPLIANCE, OBJECT-GROUP      FROM SNMPv2-CONF;

snmpFrameworkMIB MODULE-IDENTITY
    LAST-UPDATED "200210140000Z"
    ORGANIZATION "SNMPv3 Working Group"
    CONTACT-INFO "WG-EMail:   snmpv3@lists.tislabs.com
                  Subscribe:  snmpv3-request@lists.tislabs.com

                  Co-Chair:   Russ Mundy
                              Network Associates Laboratories
                  postal:     15204 Omega Drive, Suite 300
                              Rockville, MD 20850-4601
                              USA
                  EMail:      mundy@tislabs.com
                  phone:      +1 301-947-7107

                  Co-Chair &
                  Co-editor:  David Harrington
                              Enterasys Networks
                  postal:     35 Industrial Way
                              P. O. Box 5005
                              Rochester, New Hampshire 03866-5005
                              USA
                  EMail:      dbh@enterasys.com
                  phone:      +1 603-337-2614

                  Co-editor:  Randy Presuhn
                              BMC Software, Inc.
                  postal:     2141 North First Street
                              San Jose, California 95131
                              USA
                  EMail:      randy_presuhn@bmc.com
                  phone:      +1 408-546-1006

                  Co-editor:  Bert Wijnen
                              Lucent Technologies
                  postal:     Schagen 33
                              3461 GL Linschoten
                              Netherlands
```

```
                EMail:      bwijnen@lucent.com
                phone:      +31 348-680-485
                  "
      DESCRIPTION  "The SNMP Management Architecture MIB

                    Copyright (C) The Internet Society (2002). This
                    version of this MIB module is part of RFC 3411;
                    see the RFC itself for full legal notices.
                   "

      REVISION     "200210140000Z"         -- 14 October 2002
      DESCRIPTION  "Changes in this revision:
                    - Updated various administrative information.
                    - Corrected some typos.
                    - Corrected typo in description of SnmpEngineID
                      that led to range overlap for 127.
                    - Changed '255a' to '255t' in definition of
                      SnmpAdminString to align with current SMI.
                    - Reworded 'reserved' for value zero in
                      DESCRIPTION of SnmpSecurityModel.
                    - The algorithm for allocating security models
                      should give 256 per enterprise block, rather
                      than 255.
                    - The example engine ID of 'abcd' is not
                      legal. Replaced with '800002b804616263'H based
                      on example enterprise 696, string 'abc'.
                    - Added clarification that engineID should
                      persist across re-initializations.
                    This revision published as RFC 3411.
                   "
      REVISION     "199901190000Z"         -- 19 January 1999
      DESCRIPTION  "Updated editors' addresses, fixed typos.
                    Published as RFC 2571.
                   "
      REVISION     "199711200000Z"         -- 20 November 1997
      DESCRIPTION  "The initial version, published in RFC 2271.
                   "
      ::= { snmpModules 10 }

   -- Textual Conventions used in the SNMP Management Architecture ***

SnmpEngineID ::= TEXTUAL-CONVENTION
    STATUS       current
    DESCRIPTION "An SNMP engine's administratively-unique identifier.
                 Objects of this type are for identification, not for
                 addressing, even though it is possible that an
                 address may have been used in the generation of
                 a specific value.
```

The value for this object may not be all zeros or all 'ff'H or the empty (zero length) string.

The initial value for this object may be configured via an operator console entry or via an algorithmic function. In the latter case, the following example algorithm is recommended.

In cases where there are multiple engines on the same system, the use of this algorithm is NOT appropriate, as it would result in all of those engines ending up with the same ID value.

1) The very first bit is used to indicate how the rest of the data is composed.

   0 - as defined by enterprise using former methods that existed before SNMPv3. See item 2 below.

   1 - as defined by this architecture, see item 3 below.

   Note that this allows existing uses of the engineID (also known as AgentID [RFC1910]) to co-exist with any new uses.

2) The snmpEngineID has a length of 12 octets.

   The first four octets are set to the binary equivalent of the agent's SNMP management private enterprise number as assigned by the Internet Assigned Numbers Authority (IANA). For example, if Acme Networks has been assigned { enterprises 696 }, the first four octets would be assigned '000002b8'H.

   The remaining eight octets are determined via one or more enterprise-specific methods. Such methods must be designed so as to maximize the possibility that the value of this object will be unique in the agent's administrative domain. For example, it may be the IP address of the SNMP entity, or the MAC address of one of the interfaces, with each address suitably padded with random octets. If multiple methods are defined, then it is recommended that the first octet indicate the method being used and the remaining octets be a function of the method.

```
                3) The length of the octet string varies.

                   The first four octets are set to the binary
                   equivalent of the agent's SNMP management
                   private enterprise number as assigned by the
                   Internet Assigned Numbers Authority (IANA).
                   For example, if Acme Networks has been assigned
                   { enterprises 696 }, the first four octets would
                   be assigned '000002b8'H.

                   The very first bit is set to 1. For example, the
                   above value for Acme Networks now changes to be
                   '800002b8'H.

                   The fifth octet indicates how the rest (6th and
                   following octets) are formatted. The values for
                   the fifth octet are:

                     0     - reserved, unused.

                     1     - IPv4 address (4 octets)
                             lowest non-special IP address

                     2     - IPv6 address (16 octets)
                             lowest non-special IP address

                     3     - MAC address (6 octets)
                             lowest IEEE MAC address, canonical
                             order

                     4     - Text, administratively assigned
                             Maximum remaining length 27

                     5     - Octets, administratively assigned
                             Maximum remaining length 27

                     6-127 - reserved, unused

                   128-255 - as defined by the enterprise
                             Maximum remaining length 27
                "
    SYNTAX       OCTET STRING (SIZE(5..32))
```

```
SnmpSecurityModel ::= TEXTUAL-CONVENTION
    STATUS       current
    DESCRIPTION "An identifier that uniquely identifies a
                 Security Model of the Security Subsystem within
                 this SNMP Management Architecture.

                 The values for securityModel are allocated as
                 follows:

                 - The zero value does not identify any particular
                   security model.

                 - Values between 1 and 255, inclusive, are reserved
                   for standards-track Security Models and are
                   managed by the Internet Assigned Numbers Authority
                   (IANA).
                 - Values greater than 255 are allocated to
                   enterprise-specific Security Models.  An
                   enterprise-specific securityModel value is defined
                   to be:

                   enterpriseID * 256 + security model within
                   enterprise

                   For example, the fourth Security Model defined by
                   the enterprise whose enterpriseID is 1 would be
                   259.

                 This scheme for allocation of securityModel
                 values allows for a maximum of 255 standards-
                 based Security Models, and for a maximum of
                 256 Security Models per enterprise.

                 It is believed that the assignment of new
                 securityModel values will be rare in practice
                 because the larger the number of simultaneously
                 utilized Security Models, the larger the
                 chance that interoperability will suffer.
                 Consequently, it is believed that such a range
                 will be sufficient.  In the unlikely event that
                 the standards committee finds this number to be
                 insufficient over time, an enterprise number
                 can be allocated to obtain an additional 256
                 possible values.

                 Note that the most significant bit must be zero;
                 hence, there are 23 bits allocated for various
                 organizations to design and define non-standard
```

```
                securityModels.  This limits the ability to
                define new proprietary implementations of Security
                Models to the first 8,388,608 enterprises.

                It is worthwhile to note that, in its encoded
                form, the securityModel value will normally
                require only a single byte since, in practice,
                the leftmost bits will be zero for most messages
                and sign extension is suppressed by the encoding
                rules.

                As of this writing, there are several values
                of securityModel defined for use with SNMP or
                reserved for use with supporting MIB objects.
                They are as follows:

                    0  reserved for 'any'
                    1  reserved for SNMPv1
                    2  reserved for SNMPv2c
                    3  User-Based Security Model (USM)
               "
    SYNTAX      INTEGER(0 .. 2147483647)


SnmpMessageProcessingModel ::= TEXTUAL-CONVENTION
    STATUS      current
    DESCRIPTION "An identifier that uniquely identifies a Message
                Processing Model of the Message Processing
                Subsystem within this SNMP Management Architecture.

                The values for messageProcessingModel are
                allocated as follows:

                - Values between 0 and 255, inclusive, are
                  reserved for standards-track Message Processing
                  Models and are managed by the Internet Assigned
                  Numbers Authority (IANA).

                - Values greater than 255 are allocated to
                  enterprise-specific Message Processing Models.
                  An enterprise messageProcessingModel value is
                  defined to be:

                  enterpriseID * 256 +
                       messageProcessingModel within enterprise

                  For example, the fourth Message Processing Model
                  defined by the enterprise whose enterpriseID
```

```
                is 1 would be 259.

              This scheme for allocating messageProcessingModel
              values allows for a maximum of 255 standards-
              based Message Processing Models, and for a
              maximum of 256 Message Processing Models per
              enterprise.

              It is believed that the assignment of new
              messageProcessingModel values will be rare
              in practice because the larger the number of
              simultaneously utilized Message Processing Models,
              the larger the chance that interoperability
              will suffer. It is believed that such a range
              will be sufficient.  In the unlikely event that
              the standards committee finds this number to be
              insufficient over time, an enterprise number
              can be allocated to obtain an additional 256
              possible values.

              Note that the most significant bit must be zero;
              hence, there are 23 bits allocated for various
              organizations to design and define non-standard
              messageProcessingModels.  This limits the ability
              to define new proprietary implementations of
              Message Processing Models to the first 8,388,608
              enterprises.

              It is worthwhile to note that, in its encoded
              form, the messageProcessingModel value will
              normally require only a single byte since, in
              practice, the leftmost bits will be zero for
              most messages and sign extension is suppressed
              by the encoding rules.

              As of this writing, there are several values of
              messageProcessingModel defined for use with SNMP.
              They are as follows:

                  0  reserved for SNMPv1
                  1  reserved for SNMPv2c
                  2  reserved for SNMPv2u and SNMPv2*
                  3  reserved for SNMPv3
             "
    SYNTAX       INTEGER(0 .. 2147483647)
```

```
SnmpSecurityLevel ::= TEXTUAL-CONVENTION
    STATUS       current
    DESCRIPTION "A Level of Security at which SNMP messages can be
                 sent or with which operations are being processed;
                 in particular, one of:

                   noAuthNoPriv - without authentication and
                                  without privacy,
                   authNoPriv   - with authentication but
                                  without privacy,
                   authPriv     - with authentication and
                                  with privacy.

                 These three values are ordered such that
                 noAuthNoPriv is less than authNoPriv and
                 authNoPriv is less than authPriv.
                "
    SYNTAX       INTEGER { noAuthNoPriv(1),
                           authNoPriv(2),
                           authPriv(3)
                         }

SnmpAdminString ::= TEXTUAL-CONVENTION
    DISPLAY-HINT "255t"
    STATUS       current
    DESCRIPTION "An octet string containing administrative
                 information, preferably in human-readable form.

                 To facilitate internationalization, this
                 information is represented using the ISO/IEC
                 IS 10646-1 character set, encoded as an octet
                 string using the UTF-8 transformation format
                 described in [RFC2279].

                 Since additional code points are added by
                 amendments to the 10646 standard from time
                 to time, implementations must be prepared to
                 encounter any code point from 0x00000000 to
                 0x7fffffff.  Byte sequences that do not
                 correspond to the valid UTF-8 encoding of a
                 code point or are outside this range are
                 prohibited.

                 The use of control codes should be avoided.

                 When it is necessary to represent a newline,
                 the control code sequence CR LF should be used.
```

```
                The use of leading or trailing white space should
                be avoided.

                For code points not directly supported by user
                interface hardware or software, an alternative
                means of entry and display, such as hexadecimal,
                may be provided.

                For information encoded in 7-bit US-ASCII,
                the UTF-8 encoding is identical to the
                US-ASCII encoding.

                UTF-8 may require multiple bytes to represent a
                single character / code point; thus the length
                of this object in octets may be different from
                the number of characters encoded.  Similarly,
                size constraints refer to the number of encoded
                octets, not the number of characters represented
                by an encoding.

                Note that when this TC is used for an object that
                is used or envisioned to be used as an index, then
                a SIZE restriction MUST be specified so that the
                number of sub-identifiers for any object instance
                does not exceed the limit of 128, as defined by
                [RFC3416].

                Note that the size of an SnmpAdminString object is
                measured in octets, not characters.
               "
    SYNTAX       OCTET STRING (SIZE (0..255))


-- Administrative assignments ****************************************

snmpFrameworkAdmin
    OBJECT IDENTIFIER ::= { snmpFrameworkMIB 1 }
snmpFrameworkMIBObjects
    OBJECT IDENTIFIER ::= { snmpFrameworkMIB 2 }
snmpFrameworkMIBConformance
    OBJECT IDENTIFIER ::= { snmpFrameworkMIB 3 }

-- the snmpEngine Group ******************************************

snmpEngine OBJECT IDENTIFIER ::= { snmpFrameworkMIBObjects 1 }
```

```
snmpEngineID     OBJECT-TYPE
    SYNTAX       SnmpEngineID
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION "An SNMP engine's administratively-unique identifier.

                 This information SHOULD be stored in non-volatile
                 storage so that it remains constant across
                 re-initializations of the SNMP engine.
                "
    ::= { snmpEngine 1 }

snmpEngineBoots  OBJECT-TYPE
    SYNTAX       INTEGER (1..2147483647)
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION "The number of times that the SNMP engine has
                 (re-)initialized itself since snmpEngineID
                 was last configured.
                "
    ::= { snmpEngine 2 }

snmpEngineTime   OBJECT-TYPE
    SYNTAX       INTEGER (0..2147483647)
    UNITS        "seconds"
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION "The number of seconds since the value of
                 the snmpEngineBoots object last changed.
                 When incrementing this object's value would
                 cause it to exceed its maximum,
                 snmpEngineBoots is incremented as if a
                 re-initialization had occurred, and this
                 object's value consequently reverts to zero.
                "
    ::= { snmpEngine 3 }

snmpEngineMaxMessageSize OBJECT-TYPE
    SYNTAX       INTEGER (484..2147483647)
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION "The maximum length in octets of an SNMP message
                 which this SNMP engine can send or receive and
                 process, determined as the minimum of the maximum
                 message size values supported among all of the
                 transports available to and supported by the engine.
                "
    ::= { snmpEngine 4 }
```

```
-- Registration Points for Authentication and Privacy Protocols **

snmpAuthProtocols OBJECT-IDENTITY
    STATUS        current
    DESCRIPTION  "Registration point for standards-track
                  authentication protocols used in SNMP Management
                  Frameworks.
                 "
    ::= { snmpFrameworkAdmin 1 }

snmpPrivProtocols OBJECT-IDENTITY
    STATUS        current
    DESCRIPTION  "Registration point for standards-track privacy
                  protocols used in SNMP Management Frameworks.
                 "
    ::= { snmpFrameworkAdmin 2 }

-- Conformance information ****************************************

snmpFrameworkMIBCompliances
               OBJECT IDENTIFIER ::= {snmpFrameworkMIBConformance 1}
snmpFrameworkMIBGroups
               OBJECT IDENTIFIER ::= {snmpFrameworkMIBConformance 2}

-- compliance statements

snmpFrameworkMIBCompliance MODULE-COMPLIANCE
    STATUS       current
    DESCRIPTION "The compliance statement for SNMP engines which
                 implement the SNMP Management Framework MIB.
                "
    MODULE    -- this module
        MANDATORY-GROUPS { snmpEngineGroup }

    ::= { snmpFrameworkMIBCompliances 1 }

-- units of conformance

snmpEngineGroup OBJECT-GROUP
    OBJECTS {
              snmpEngineID,
              snmpEngineBoots,
              snmpEngineTime,
              snmpEngineMaxMessageSize
            }
    STATUS       current
    DESCRIPTION "A collection of objects for identifying and
                 determining the configuration and current timeliness
```

```
                values of an SNMP engine.
              "
    ::= { snmpFrameworkMIBGroups 1 }

END
```

6. IANA Considerations

This document defines three number spaces administered by IANA, one for security models, another for message processing models, and a third for SnmpEngineID formats.

6.1. Security Models

The SnmpSecurityModel TEXTUAL-CONVENTION values managed by IANA are in the range from 0 to 255 inclusive, and are reserved for standards-track Security Models. If this range should in the future prove insufficient, an enterprise number can be allocated to obtain an additional 256 possible values.

As of this writing, there are several values of securityModel defined for use with SNMP or reserved for use with supporting MIB objects. They are as follows:

```
0  reserved for 'any'
1  reserved for SNMPv1
2  reserved for SNMPv2c
3  User-Based Security Model (USM)
```

6.2. Message Processing Models

The SnmpMessageProcessingModel TEXTUAL-CONVENTION values managed by IANA are in the range 0 to 255, inclusive. Each value uniquely identifies a standards-track Message Processing Model of the Message Processing Subsystem within the SNMP Management Architecture.

Should this range prove insufficient in the future, an enterprise number may be obtained for the standards committee to get an additional 256 possible values.

As of this writing, there are several values of messageProcessingModel defined for use with SNMP. They are as follows:

```
0  reserved for SNMPv1
1  reserved for SNMPv2c
2  reserved for SNMPv2u and SNMPv2*
3  reserved for SNMPv3
```

6.3. SnmpEngineID Formats

The SnmpEngineID TEXTUAL-CONVENTION's fifth octet contains a format identifier. The values managed by IANA are in the range 6 to 127, inclusive. Each value uniquely identifies a standards-track SnmpEngineID format.

7. Intellectual Property

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-related documentation can be found in RFC 2028. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to 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.

8. Acknowledgements

This document is the result of the efforts of the SNMPv3 Working Group. Some special thanks are in order to the following SNMPv3 WG members:

Harald Tveit Alvestrand (Maxware)
Dave Battle (SNMP Research, Inc.)
Alan Beard (Disney Worldwide Services)
Paul Berrevoets (SWI Systemware/Halcyon Inc.)
Martin Bjorklund (Ericsson)
Uri Blumenthal (IBM T.J. Watson Research Center)
Jeff Case (SNMP Research, Inc.)
John Curran (BBN)
Mike Daniele (Compaq Computer Corporation)
T. Max Devlin (Eltrax Systems)
John Flick (Hewlett Packard)
Rob Frye (MCI)
Wes Hardaker (U.C.Davis, Information Technology - D.C.A.S.)

David Harrington (Cabletron Systems Inc.)
Lauren Heintz (BMC Software, Inc.)
N.C. Hien (IBM T.J. Watson Research Center)
Michael Kirkham (InterWorking Labs, Inc.)
Dave Levi (SNMP Research, Inc.)
Louis A Mamakos (UUNET Technologies Inc.)
Joe Marzot (Nortel Networks)
Paul Meyer (Secure Computing Corporation)
Keith McCloghrie (Cisco Systems)
Bob Moore (IBM)
Russ Mundy (TIS Labs at Network Associates)
Bob Natale (ACE*COMM Corporation)
Mike O'Dell (UUNET Technologies Inc.)
Dave Perkins (DeskTalk)
Peter Polkinghorne (Brunel University)
Randy Presuhn (BMC Software, Inc.)
David Reeder (TIS Labs at Network Associates)
David Reid (SNMP Research, Inc.)
Aleksey Romanov (Quality Quorum)
Shawn Routhier (Epilogue)
Juergen Schoenwaelder (TU Braunschweig)
Bob Stewart (Cisco Systems)
Mike Thatcher (Independent Consultant)
Bert Wijnen (IBM T.J. Watson Research Center)

The document is based on recommendations of the IETF Security and Administrative Framework Evolution for SNMP Advisory Team. Members of that Advisory Team were:

David Harrington (Cabletron Systems Inc.)
Jeff Johnson (Cisco Systems)
David Levi (SNMP Research Inc.)
John Linn (Openvision)
Russ Mundy (Trusted Information Systems) chair
Shawn Routhier (Epilogue)
Glenn Waters (Nortel)
Bert Wijnen (IBM T. J. Watson Research Center)

As recommended by the Advisory Team and the SNMPv3 Working Group Charter, the design incorporates as much as practical from previous RFCs and drafts. As a result, special thanks are due to the authors of previous designs known as SNMPv2u and SNMPv2*:

Jeff Case (SNMP Research, Inc.)
David Harrington (Cabletron Systems Inc.)
David Levi (SNMP Research, Inc.)
Keith McCloghrie (Cisco Systems)
Brian O'Keefe (Hewlett Packard)

Marshall T. Rose (Dover Beach Consulting)
Jon Saperia (BGS Systems Inc.)
Steve Waldbusser (International Network Services)
Glenn W. Waters (Bell-Northern Research Ltd.)

9. Security Considerations

This document describes how an implementation can include a Security Model to protect management messages and an Access Control Model to control access to management information.

The level of security provided is determined by the specific Security Model implementation(s) and the specific Access Control Model implementation(s) used.

Applications have access to data which is not secured. Applications SHOULD take reasonable steps to protect the data from disclosure.

It is the responsibility of the purchaser of an implementation to ensure that:

1) an implementation complies with the rules defined by this architecture,

2) the Security and Access Control Models utilized satisfy the security and access control needs of the organization,

3) the implementations of the Models and Applications comply with the model and application specifications,

4) and the implementation protects configuration secrets from inadvertent disclosure.

This document also contains a MIB definition module. None of the objects defined is writable, and the information they represent is not deemed to be particularly sensitive. However, if they are deemed sensitive in a particular environment, access to them should be restricted through the use of appropriately configured Security and Access Control models.

10. References

10.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2279] Yergeau, F., "UTF-8, a transformation format of ISO 10646", RFC 2279, January 1998.

[RFC2578] McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M. and S. Waldbusser, "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, April 1999.

[RFC2579] McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M. and S. Waldbusser, "Textual Conventions for SMIv2", STD 58, RFC 2579, April 1999.

[RFC2580] McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M. and S. Waldbusser, "Conformance Statements for SMIv2", STD 58, RFC 2580, April 1999.

[RFC3412] 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.

[RFC3413] Levi, D., Meyer, P. and B. Stewart, "Simple Network Management Protocol (SNMP) Applications", STD 62, RFC 3413, December 2002.

[RFC3414] 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.

[RFC3415] 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.

[RFC3416] Presuhn, R., Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Protocol Operations for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3416, December 2002.

[RFC3417] Presuhn, R., Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Transport Mappings for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3417, December 2002.

[RFC3418] Presuhn, R., Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Management Information Base (MIB) for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3418, December 2002.

10.2. Informative References

[RFC1155] Rose, M. and K. McCloghrie, "Structure and Identification of Management Information for TCP/IP-based internets", STD 16, RFC 1155, May 1990.

[RFC1157] Case, J., Fedor, M., Schoffstall, M. and J. Davin, "The Simple Network Management Protocol", STD 15, RFC 1157, May 1990.

[RFC1212] Rose, M. and K. McCloghrie, "Concise MIB Definitions", STD 16, RFC 1212, March 1991.

[RFC1901] Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Introduction to Community-based SNMPv2", RFC 1901, January 1996.

[RFC1909] McCloghrie, K., Editor, "An Administrative Infrastructure for SNMPv2", RFC 1909, February 1996.

[RFC1910] Waters, G., Editor, "User-based Security Model for SNMPv2", RFC 1910, February 1996.

[RFC2028] Hovey, R. and S. Bradner, "The Organizations Involved in the IETF Standards Process", BCP 11, RFC 2028, October 1996.

[RFC2576] Frye, R., Levi, D., Routhier, S. and B. Wijnen, "Coexistence between Version 1, Version 2, and Version 3 of the Internet-Standard Network Management Framework", RFC 2576, March 2000.

[RFC2863] McCloghrie, K. and F. Kastenholz, "The Interfaces Group MIB", RFC 2863, June 2000.

[RFC3410] Case, J., Mundy, R., Partain, D. and B. Stewart, "Introduction and Applicability Statements for Internet-Standard Management Framework", RFC 3410, December 2002.

Appendix A

A. Guidelines for Model Designers

This appendix describes guidelines for designers of models which are expected to fit into the architecture defined in this document.

SNMPv1 and SNMPv2c are two SNMP frameworks which use communities to provide trivial authentication and access control. SNMPv1 and SNMPv2c Frameworks can coexist with Frameworks designed according to this architecture, and modified versions of SNMPv1 and SNMPv2c Frameworks could be designed to meet the requirements of this architecture, but this document does not provide guidelines for that coexistence.

Within any subsystem model, there should be no reference to any specific model of another subsystem, or to data defined by a specific model of another subsystem.

Transfer of data between the subsystems is deliberately described as a fixed set of abstract data elements and primitive functions which can be overloaded to satisfy the needs of multiple model definitions.

Documents which define models to be used within this architecture SHOULD use the standard primitives between subsystems, possibly defining specific mechanisms for converting the abstract data elements into model-usable formats. This constraint exists to allow subsystem and model documents to be written recognizing common borders of the subsystem and model. Vendors are not constrained to recognize these borders in their implementations.

The architecture defines certain standard services to be provided between subsystems, and the architecture defines abstract service interfaces to request these services.

Each model definition for a subsystem SHOULD support the standard service interfaces, but whether, or how, or how well, it performs the service is dependent on the model definition.

A.1. Security Model Design Requirements

A.1.1. Threats

A document describing a Security Model MUST describe how the model protects against the threats described under "Security Requirements of this Architecture", section 1.4.

A.1.2. Security Processing

Received messages MUST be validated by a Model of the Security Subsystem. Validation includes authentication and privacy processing if needed, but it is explicitly allowed to send messages which do not require authentication or privacy.

A received message contains a specified securityLevel to be used during processing. All messages requiring privacy MUST also require authentication.

A Security Model specifies rules by which authentication and privacy are to be done. A model may define mechanisms to provide additional security features, but the model definition is constrained to using (possibly a subset of) the abstract data elements defined in this document for transferring data between subsystems.

Each Security Model may allow multiple security protocols to be used concurrently within an implementation of the model. Each Security Model defines how to determine which protocol to use, given the securityLevel and the security parameters relevant to the message. Each Security Model, with its associated protocol(s) defines how the sending/receiving entities are identified, and how secrets are configured.

Authentication and Privacy protocols supported by Security Models are uniquely identified using Object Identifiers. IETF standard protocols for authentication or privacy should have an identifier defined within the snmpAuthProtocols or the snmpPrivProtocols subtrees. Enterprise specific protocol identifiers should be defined within the enterprise subtree.

For privacy, the Security Model defines what portion of the message is encrypted.

The persistent data used for security should be SNMP-manageable, but the Security Model defines whether an instantiation of the MIB is a conformance requirement.

Security Models are replaceable within the Security Subsystem. Multiple Security Model implementations may exist concurrently within an SNMP engine. The number of Security Models defined by the SNMP community should remain small to promote interoperability.

A.1.3. Validate the security-stamp in a received message

A Message Processing Model requests that a Security Model:

- verifies that the message has not been altered,
- authenticates the identification of the principal for whom the message was generated.
- decrypts the message if it was encrypted.

Additional requirements may be defined by the model, and additional services may be provided by the model, but the model is constrained to use the following primitives for transferring data between subsystems. Implementations are not so constrained.

A Message Processing Model uses the processIncomingMsg primitive as described in section 4.4.2.

A.1.4. Security MIBs

Each Security Model defines the MIB module(s) required for security processing, including any MIB module(s) required for the security protocol(s) supported. The MIB module(s) SHOULD be defined concurrently with the procedures which use the MIB module(s). The MIB module(s) are subject to normal access control rules.

The mapping between the model-dependent security ID and the securityName MUST be able to be determined using SNMP, if the model-dependent MIB is instantiated and if access control policy allows access.

A.1.5. Cached Security Data

For each message received, the Security Model caches the state information such that a Response message can be generated using the same security information, even if the Local Configuration Datastore is altered between the time of the incoming request and the outgoing response.

A Message Processing Model has the responsibility for explicitly releasing the cached data if such data is no longer needed. To enable this, an abstract securityStateReference data element is passed from the Security Model to the Message Processing Model.

The cached security data may be implicitly released via the generation of a response, or explicitly released by using the stateRelease primitive, as described in section 4.5.1.

A.2. Message Processing Model Design Requirements

An SNMP engine contains a Message Processing Subsystem which may contain multiple Message Processing Models.

The Message Processing Model MUST always (conceptually) pass the complete PDU, i.e., it never forwards less than the complete list of varBinds.

A.2.1. Receiving an SNMP Message from the Network

Upon receipt of a message from the network, the Dispatcher in the SNMP engine determines the version of the SNMP message and interacts with the corresponding Message Processing Model to determine the abstract data elements.

A Message Processing Model specifies the SNMP Message format it supports and describes how to determine the values of the abstract data elements (like msgID, msgMaxSize, msgFlags, msgSecurityParameters, securityModel, securityLevel etc). A Message Processing Model interacts with a Security Model to provide security processing for the message using the processIncomingMsg primitive, as described in section 4.4.2.

A.2.2. Sending an SNMP Message to the Network

The Dispatcher in the SNMP engine interacts with a Message Processing Model to prepare an outgoing message. For that it uses the following primitives:

- for requests and notifications: prepareOutgoingMessage, as described in section 4.2.1.

- for response messages: prepareResponseMessage, as described in section 4.2.2.

A Message Processing Model, when preparing an Outgoing SNMP Message, interacts with a Security Model to secure the message. For that it uses the following primitives:

- for requests and notifications: generateRequestMsg, as described in section 4.4.1.

- for response messages: generateResponseMsg as described in section 4.4.3.

Once the SNMP message is prepared by a Message Processing Model, the Dispatcher sends the message to the desired address using the appropriate transport.

A.3. Application Design Requirements

Within an application, there may be an explicit binding to a specific SNMP message version, i.e., a specific Message Processing Model, and to a specific Access Control Model, but there should be no reference to any data defined by a specific Message Processing Model or Access Control Model.

Within an application, there should be no reference to any specific Security Model, or any data defined by a specific Security Model.

An application determines whether explicit or implicit access control should be applied to the operation, and, if access control is needed, which Access Control Model should be used.

An application has the responsibility to define any MIB module(s) used to provide application-specific services.

Applications interact with the SNMP engine to initiate messages, receive responses, receive asynchronous messages, and send responses.

A.3.1. Applications that Initiate Messages

Applications may request that the SNMP engine send messages containing SNMP commands or notifications using the sendPdu primitive as described in section 4.1.1.

If it is desired that a message be sent to multiple targets, it is the responsibility of the application to provide the iteration.

The SNMP engine assumes necessary access control has been applied to the PDU, and provides no access control services.

The SNMP engine looks at the "expectResponse" parameter, and if a response is expected, then the appropriate information is cached such that a later response can be associated to this message, and can then be returned to the application. A sendPduHandle is returned to the application so it can later correspond the response with this message as well.

A.3.2. Applications that Receive Responses

The SNMP engine matches the incoming response messages to outstanding messages sent by this SNMP engine, and forwards the response to the associated application using the processResponsePdu primitive, as described in section 4.1.4.

A.3.3. Applications that Receive Asynchronous Messages

When an SNMP engine receives a message that is not the response to a request from this SNMP engine, it must determine to which application the message should be given.

An Application that wishes to receive asynchronous messages registers itself with the engine using the primitive registerContextEngineID as described in section 4.1.5.

An Application that wishes to stop receiving asynchronous messages should unregister itself with the SNMP engine using the primitive unregisterContextEngineID as described in section 4.1.5.

Only one registration per combination of PDU type and contextEngineID is permitted at the same time. Duplicate registrations are ignored. An errorIndication will be returned to the application that attempts to duplicate a registration.

All asynchronously received messages containing a registered combination of PDU type and contextEngineID are sent to the application which registered to support that combination.

The engine forwards the PDU to the registered application, using the processPdu primitive, as described in section 4.1.2.

A.3.4. Applications that Send Responses

Request operations require responses. An application sends a response via the returnResponsePdu primitive, as described in section 4.1.3.

The contextEngineID, contextName, securityModel, securityName, securityLevel, and stateReference parameters are from the initial processPdu primitive. The PDU and statusInformation are the results of processing.

A.4. Access Control Model Design Requirements

An Access Control Model determines whether the specified securityName is allowed to perform the requested operation on a specified managed object. The Access Control Model specifies the rules by which access control is determined.

The persistent data used for access control should be manageable using SNMP, but the Access Control Model defines whether an instantiation of the MIB is a conformance requirement.

The Access Control Model must provide the primitive isAccessAllowed.

Editors' Addresses

Bert Wijnen
Lucent Technologies
Schagen 33
3461 GL Linschoten
Netherlands

Phone: +31 348-680-485
EMail: bwijnen@lucent.com

David Harrington
Enterasys Networks
Post Office Box 5005
35 Industrial Way
Rochester, New Hampshire 03866-5005
USA

Phone: +1 603-337-2614
EMail: dbh@enterasys.com

Randy Presuhn
BMC Software, Inc.
2141 North First Street
San Jose, California 95131
USA

Phone: +1 408-546-1006
Fax: +1 408-965-0359
EMail: randy_presuhn@bmc.com

Full Copyright Statement

Acknowledgement

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

Network Working Group
Request for Comments: 3412
STD: 62
Obsoletes: 2572
Category: Standards Track

J. Case
SNMP Research, Inc.
D. Harrington
Enterasys Networks
R. Presuhn
BMC Software, Inc.
B. Wijnen
Lucent Technologies

December 2002

Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)

Status of this Memo

Copyright Notice

Abstract

This document describes the Message Processing and Dispatching for Simple Network Management Protocol (SNMP) messages within the SNMP architecture. It defines the procedures for dispatching potentially multiple versions of SNMP messages to the proper SNMP Message Processing Models, and for dispatching PDUs to SNMP applications. This document also describes one Message Processing Model - the SNMPv3 Message Processing Model. This document obsoletes RFC 2572.

Table of Contents

1. Introduction

The Architecture for describing Internet Management Frameworks [RFC3411] describes that an SNMP engine is composed of:

1) a Dispatcher
2) a Message Processing Subsystem,
3) a Security Subsystem, and
4) an Access Control Subsystem.

Applications make use of the services of these subsystems.

It is important to understand the SNMP architecture and its terminology to understand where the Message Processing Subsystem and Dispatcher described in this document fit into the architecture and interact with other subsystems within the architecture. The reader is expected to have read and understood the description of the SNMP architecture, defined in [RFC3411].

The Dispatcher in the SNMP engine sends and receives SNMP messages. It also dispatches SNMP PDUs to SNMP applications. When an SNMP message needs to be prepared or when data needs to be extracted from an SNMP message, the Dispatcher delegates these tasks to a message version-specific Message Processing Model within the Message Processing Subsystem.

A Message Processing Model is responsible for processing an SNMP version-specific message and for coordinating the interaction with the Security Subsystem to ensure proper security is applied to the SNMP message being handled.

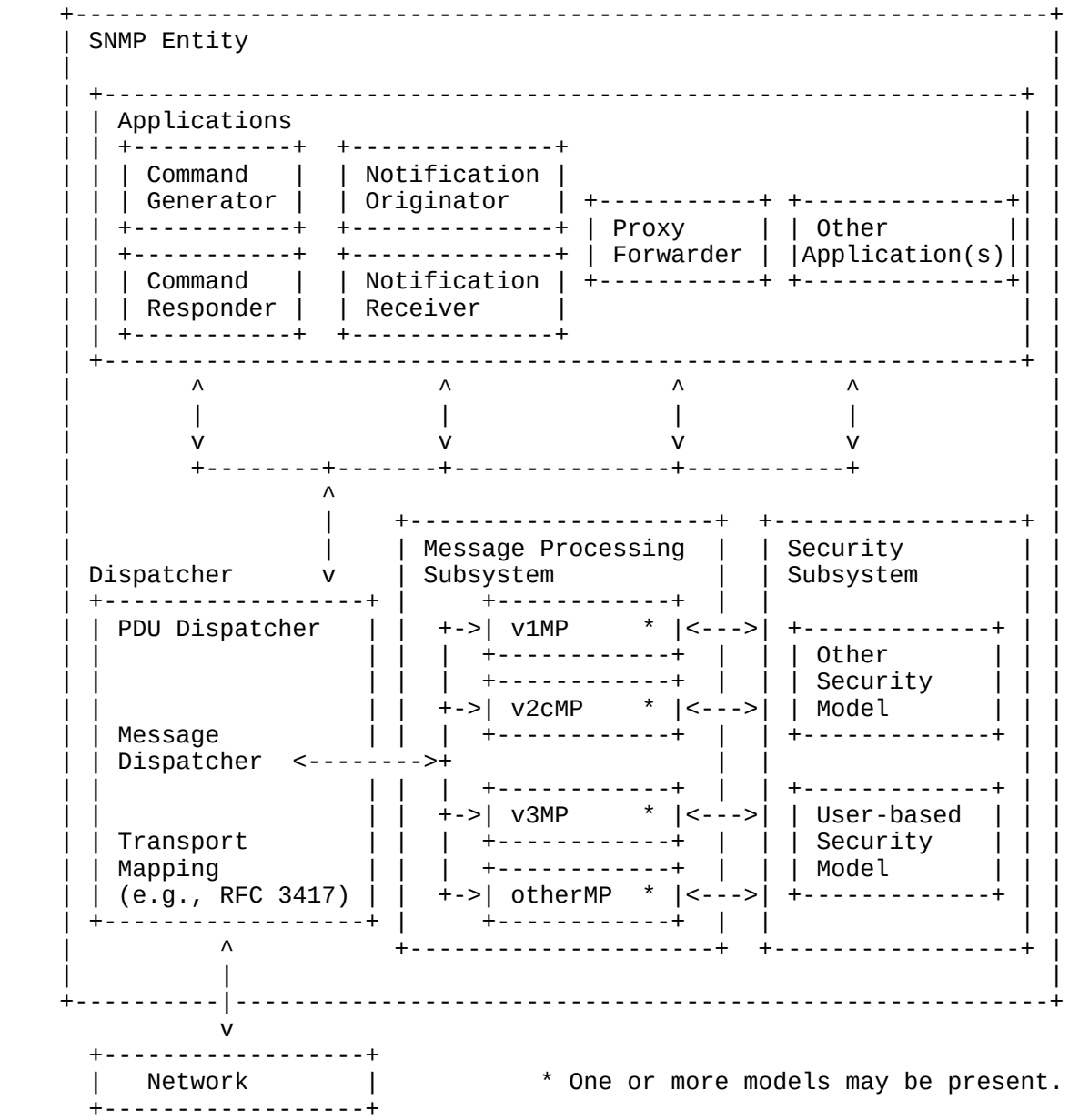
Interactions between the Dispatcher, the Message Processing Subsystem, and applications are modeled using abstract data elements and abstract service interface primitives defined by the SNMP architecture.

Similarly, interactions between the Message Processing Subsystem and the Security Subsystem are modeled using abstract data elements and abstract service interface primitives as defined by the SNMP architecture.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, RFC 2119.

2. Overview

The following illustration depicts the Message Processing in relation to SNMP applications, the Security Subsystem and Transport Mappings.



2.1. The Dispatcher

The Dispatcher is a key piece of an SNMP engine. There is only one in an SNMP engine, and its job is to dispatch tasks to the multiple version-specific Message Processing Models, and to dispatch PDUs to various applications.

For outgoing messages, an application provides a PDU to be sent, plus the data needed to prepare and send the message, and the application specifies which version-specific Message Processing Model will be used to prepare the message with the desired security processing. Once the message is prepared, the Dispatcher sends the message.

For incoming messages, the Dispatcher determines the SNMP version of the incoming message and passes the message to the version-specific Message Processing Model to extract the components of the message and to coordinate the processing of security services for the message. After version-specific processing, the PDU Dispatcher determines which application, if any, should receive the PDU for processing and forwards it accordingly.

The Dispatcher, while sending and receiving SNMP messages, collects statistics about SNMP messages and the behavior of the SNMP engine in managed objects to make them accessible to remote SNMP entities. This document defines these managed objects, the MIB module which contains them, and how these managed objects might be used to provide useful management.

2.2. Message Processing Subsystem

The SNMP Message Processing Subsystem is the part of an SNMP engine which interacts with the Dispatcher to handle the version-specific SNMP messages. It contains one or more Message Processing Models.

This document describes one Message Processing Model, the SNMPv3 Message Processing Model, in Section 6. The SNMPv3 Message Processing Model is defined in a separate section to show that multiple (independent) Message Processing Models can exist at the same time and that such Models can be described in different documents. The SNMPv3 Message Processing Model can be replaced or supplemented with other Message Processing Models in the future. Two Message Processing Models which are expected to be developed in the future are the SNMPv1 message format [RFC1157] and the SNMPv2c message format [RFC1901]. Others may be developed as needed.

3. Elements of Message Processing and Dispatching

See [RFC3411] for the definitions of:

- contextEngineID
- contextName
- scopedPDU
- maxSizeResponseScopedPDU
- securityModel
- securityName
- securityLevel
- messageProcessingModel

For incoming messages, a version-specific message processing module provides these values to the Dispatcher. For outgoing messages, an application provides these values to the Dispatcher.

For some version-specific processing, the values may be extracted from received messages; for other versions, the values may be determined by algorithm, or by an implementation-defined mechanism. The mechanism by which the value is determined is irrelevant to the Dispatcher.

The following additional or expanded definitions are for use within the Dispatcher.

3.1. messageProcessingModel

The value of messageProcessingModel identifies a Message Processing Model. A Message Processing Model describes the version-specific procedures for extracting data from messages, generating messages, calling upon a securityModel to apply its security services to messages, for converting data from a version-specific message format into a generic format usable by the Dispatcher, and for converting data from Dispatcher format into a version-specific message format.

3.2. pduVersion

The value of pduVersion represents a specific version of protocol operation and its associated PDU formats, such as SNMPv1 or SNMPv2 [RFC3416]. The values of pduVersion are specific to the version of the PDU contained in a message, and the PDUs processed by applications. The Dispatcher does not use the value of pduVersion directly.

An application specifies the pduVersion when it requests the PDU Dispatcher to send a PDU to another SNMP engine. The Dispatcher passes the pduVersion to a Message Processing Model, so it knows how to handle the PDU properly.

For incoming messages, the pduVersion is provided to the Dispatcher by a version-specific Message Processing module. The PDU Dispatcher passes the pduVersion to the application so it knows how to handle the PDU properly. For example, a command responder application needs to know whether to use [RFC3416] elements of procedure and syntax instead of those specified for SNMPv1.

3.3. pduType

A value of the pduType represents a specific type of protocol operation. The values of the pduType are specific to the version of the PDU contained in a message.

Applications register to support particular pduTypes for particular contextEngineIDs.

For incoming messages, pduType is provided to the Dispatcher by a version-specific Message Processing module. It is subsequently used to dispatch the PDU to the application which registered for the pduType for the contextEngineID of the associated scopedPDU.

3.4. sendPduHandle

This handle is generated for coordinating the processing of requests and responses between the SNMP engine and an application. The handle must be unique across all version-specific Message Processing Models, and is of local significance only.

4. Dispatcher Elements of Procedure

This section describes the procedures followed by the Dispatcher when generating and processing SNMP messages.

4.1. Sending an SNMP Message to the Network

This section describes the procedure followed by an SNMP engine whenever it sends an SNMP message.

4.1.1. Sending a Request or Notification

The following procedures are followed by the Dispatcher when an application wants to send an SNMP PDU to another (remote) application, i.e., to initiate a communication by originating a message, such as one containing a request or a notification.

1) The application requests this using the abstract service primitive:

```
statusInformation =              -- sendPduHandle if success
                                 -- errorIndication if failure
  sendPdu(
  IN   transportDomain           -- transport domain to be used
  IN   transportAddress          -- destination network address
  IN   messageProcessingModel    -- typically, SNMP version
  IN   securityModel             -- Security Model to use
  IN   securityName              -- on behalf of this principal
  IN   securityLevel             -- Level of Security requested
  IN   contextEngineID           -- data from/at this entity
  IN   contextName               -- data from/in this context
  IN   pduVersion                -- the version of the PDU
  IN   PDU                       -- SNMP Protocol Data Unit
  IN   expectResponse            -- TRUE or FALSE
       )
```

2) If the messageProcessingModel value does not represent a Message Processing Model known to the Dispatcher, then an errorIndication (implementation-dependent) is returned to the calling application. No further processing is performed.

3) The Dispatcher generates a sendPduHandle to coordinate subsequent processing.

4) The Message Dispatcher sends the request to the version-specific Message Processing module identified by messageProcessingModel using the abstract service primitive:

```
statusInformation =              -- success or error indication
  prepareOutgoingMessage(
  IN   transportDomain           -- as specified by application
  IN   transportAddress          -- as specified by application
  IN   messageProcessingModel    -- as specified by application
  IN   securityModel             -- as specified by application
  IN   securityName              -- as specified by application
  IN   securityLevel             -- as specified by application
  IN   contextEngineID           -- as specified by application
  IN   contextName               -- as specified by application
  IN   pduVersion                -- as specified by application
  IN   PDU                       -- as specified by application
  IN   expectResponse            -- as specified by application
  IN   sendPduHandle             -- as determined in step 3.
  OUT  destTransportDomain       -- destination transport domain
  OUT  destTransportAddress      -- destination transport address
  OUT  outgoingMessage           -- the message to send
  OUT  outgoingMessageLength     -- the message length
       )
```

5) If the statusInformation indicates an error, the errorIndication is returned to the calling application. No further processing is performed.

6) If the statusInformation indicates success, the sendPduHandle is returned to the application, and the outgoingMessage is sent. The transport used to send the outgoingMessage is returned via destTransportDomain, and the address to which it was sent is returned via destTransportAddress.

Outgoing Message Processing is complete.

4.1.2. Sending a Response to the Network

The following procedure is followed when an application wants to return a response back to the originator of an SNMP Request.

1) An application can request this using the abstract service primitive:

```
result =
returnResponsePdu(
 IN   messageProcessingModel   -- typically, SNMP version
 IN   securityModel            -- Security Model in use
 IN   securityName             -- on behalf of this principal
 IN   securityLevel            -- same as on incoming request
 IN   contextEngineID          -- data from/at this SNMP entity
 IN   contextName              -- data from/in this context
 IN   pduVersion               -- the version of the PDU
 IN   PDU                      -- SNMP Protocol Data Unit
 IN   maxSizeResponseScopedPDU -- maximum size of Response PDU
 IN   stateReference           -- reference to state information
                               -- as presented with the request
 IN   statusInformation        -- success or errorIndication
 )                             -- (error counter OID and value
                               -- when errorIndication)
```

2) The Message Dispatcher sends the request to the appropriate Message Processing Model indicated by the received value of messageProcessingModel using the abstract service primitive:

```
result =                       -- SUCCESS or errorIndication
 prepareResponseMessage(
 IN   messageProcessingModel   -- specified by application
 IN   securityModel            -- specified by application
 IN   securityName             -- specified by application
 IN   securityLevel            -- specified by application
 IN   contextEngineID          -- specified by application
 IN   contextName              -- specified by application
 IN   pduVersion               -- specified by application
 IN   PDU                      -- specified by application
 IN   maxSizeResponseScopedPDU -- specified by application
 IN   stateReference           -- specified by application
 IN   statusInformation        -- specified by application
 OUT  destTransportDomain      -- destination transport domain
 OUT  destTransportAddress     -- destination transport address
 OUT  outgoingMessage          -- the message to send
 OUT  outgoingMessageLength    -- the message length
      )
```

3) If the result is an errorIndication, the errorIndication is returned to the calling application. No further processing is performed.

4) If the result is success, the outgoingMessage is sent. The transport used to send the outgoingMessage is returned via destTransportDomain, and the address to which it was sent is returned via destTransportAddress.

Message Processing is complete.

4.2. Receiving an SNMP Message from the Network

This section describes the procedure followed by an SNMP engine whenever it receives an SNMP message.

Please note, that for the sake of clarity and to prevent the text from being even longer and more complicated, some details were omitted from the steps below. In particular, the elements of procedure do not always explicitly indicate when state information needs to be released. The general rule is that if state information is available when a message is to be "discarded without further processing", then the state information must also be released at that same time.

4.2.1. Message Dispatching of received SNMP Messages

1) The snmpInPkts counter [RFC3418] is incremented.

2) The version of the SNMP message is determined in an implementation-dependent manner. If the packet cannot be sufficiently parsed to determine the version of the SNMP message, then the snmpInASNParseErrs [RFC3418] counter is incremented, and the message is discarded without further processing. If the version is not supported, then the snmpInBadVersions [RFC3418] counter is incremented, and the message is discarded without further processing.

3) The origin transportDomain and origin transportAddress are determined.

4) The message is passed to the version-specific Message Processing Model which returns the abstract data elements required by the Dispatcher. This is performed using the abstract service primitive:

```
result =                          -- SUCCESS or errorIndication
  prepareDataElements(
  IN   transportDomain            -- origin as determined in step 3.
  IN   transportAddress           -- origin as determined in step 3.
  IN   wholeMsg                   -- as received from the network
  IN   wholeMsgLength             -- as received from the network
  OUT  messageProcessingModel     -- typically, SNMP version
  OUT  securityModel              -- Security Model specified
  OUT  securityName               -- on behalf of this principal
  OUT  securityLevel              -- Level of Security specified
  OUT  contextEngineID            -- data from/at this entity
  OUT  contextName                -- data from/in this context
  OUT  pduVersion                 -- the version of the PDU
  OUT  PDU                        -- SNMP Protocol Data Unit
  OUT  pduType                    -- SNMP PDU type
  OUT  sendPduHandle              -- handle for a matched request
  OUT  maxSizeResponseScopedPDU -- maximum size of Response PDU
  OUT  statusInformation          -- success or errorIndication
                                  -- (error counter OID and value
                                  -- when errorIndication)
  OUT  stateReference             -- reference to state information
                                  -- to be used for a possible
       )                          -- Response
```

5) If the result is a FAILURE errorIndication, the message is discarded without further processing.

6) At this point, the abstract data elements have been prepared and processing continues as described in Section 4.2.2, PDU Dispatching for Incoming Messages.

4.2.2. PDU Dispatching for Incoming Messages

The elements of procedure for the dispatching of PDUs depends on the value of sendPduHandle. If the value of sendPduHandle is <none>, then this is a request or notification and the procedures specified in Section 4.2.2.1 apply. If the value of snmpPduHandle is not <none>, then this is a response and the procedures specified in Section 4.2.2.2 apply.

4.2.2.1. Incoming Requests and Notifications

The following procedures are followed for the dispatching of PDUs when the value of sendPduHandle is <none>, indicating this is a request or notification.

1) The combination of contextEngineID and pduType is used to determine which application has registered for this request or notification.

2) If no application has registered for the combination, then:

   a) The snmpUnknownPDUHandlers counter is incremented.

   b) A Response message is generated using the abstract service primitive:

```
result =                        -- SUCCESS or FAILURE
prepareResponseMessage(
IN   messageProcessingModel     -- as provided by MP module
IN   securityModel              -- as provided by MP module
IN   securityName               -- as provided by MP module
IN   securityLevel              -- as provided by MP module
IN   contextEngineID            -- as provided by MP module
IN   contextName                -- as provided by MP module
IN   pduVersion                 -- as provided by MP module
IN   PDU                        -- as provided by MP module
IN   maxSizeResponseScopedPDU   -- as provided by MP module
IN   stateReference             -- as provided by MP module
IN   statusInformation          -- errorIndication plus
                                -- snmpUnknownPDUHandlers OID
                                -- value pair.
OUT  destTransportDomain        -- destination transportDomain
OUT  destTransportAddress       -- destination transportAddress
OUT  outgoingMessage            -- the message to send
OUT  outgoingMessageLength      -- its length
)
```

   c) If the result is SUCCESS, then the prepared message is sent to the originator of the request as identified by the transportDomain and transportAddress. The transport used to send the outgoingMessage is returned via destTransportDomain, and the address to which it was sent is returned via destTransportAddress.

   d) The incoming message is discarded without further processing. Message Processing for this message is complete.

3) The PDU is dispatched to the application, using the abstract service primitive:

```
processPdu(                       -- process Request/Notification
  IN   messageProcessingModel     -- as provided by MP module
  IN   securityModel              -- as provided by MP module
  IN   securityName               -- as provided by MP module
  IN   securityLevel              -- as provided by MP module
  IN   contextEngineID            -- as provided by MP module
  IN   contextName                -- as provided by MP module
  IN   pduVersion                 -- as provided by MP module
  IN   PDU                        -- as provided by MP module
  IN   maxSizeResponseScopedPDU   -- as provided by MP module
  IN   stateReference             -- as provided by MP module
                                  -- needed when sending response
       )
```

Message processing for this message is complete.

4.2.2.2. Incoming Responses

The following procedures are followed for the dispatching of PDUs when the value of sendPduHandle is not <none>, indicating this is a response.

1) The value of sendPduHandle is used to determine, in an implementation-defined manner, which application is waiting for a response associated with this sendPduHandle.

2) If no waiting application is found, the message is discarded without further processing, and the stateReference is released. The snmpUnknownPDUHandlers counter is incremented. Message Processing is complete for this message.

3) Any cached information, including stateReference, about the message is discarded.

4) The response is dispatched to the application using the abstract service primitive:

```
processResponsePdu(              -- process Response PDU
  IN   messageProcessingModel    -- provided by the MP module
  IN   securityModel             -- provided by the MP module
  IN   securityName              -- provided by the MP module
  IN   securityLevel             -- provided by the MP module
  IN   contextEngineID           -- provided by the MP module
  IN   contextName               -- provided by the MP module
  IN   pduVersion                -- provided by the MP module
  IN   PDU                       -- provided by the MP module
  IN   statusInformation         -- provided by the MP module
  IN   sendPduHandle             -- provided by the MP module
       )
```

Message Processing is complete for this message.

4.3. Application Registration for Handling PDU types

Applications that want to process certain PDUs must register with the PDU Dispatcher. Applications specify the combination of contextEngineID and pduType(s) for which they want to take responsibility.

1) An application registers according to the abstract interface primitive:

```
statusInformation =          -- success or errorIndication
  registerContextEngineID(
  IN   contextEngineID       -- take responsibility for this one
  IN   pduType               -- the pduType(s) to be registered
       )
```

Note: Implementations may provide a means of requesting registration for simultaneous multiple contextEngineID values, e.g., all contextEngineID values, and may also provide a means for requesting simultaneous registration for multiple values of the pduType.

2) The parameters may be checked for validity; if they are not, then an errorIndication (invalidParameter) is returned to the application.

3) Each combination of contextEngineID and pduType can be registered only once. If another application has already registered for the specified combination, then an errorIndication (alreadyRegistered) is returned to the application.

4) Otherwise, the registration is saved so that SNMP PDUs can be dispatched to this application.

4.4. Application Unregistration for Handling PDU Types

Applications that no longer want to process certain PDUs must unregister with the PDU Dispatcher.

1) An application unregisters using the abstract service primitive:

```
unregisterContextEngineID(
 IN   contextEngineID        -- give up responsibility for this
 IN   pduType                -- the pduType(s) to be unregistered
      )
```

Note: Implementations may provide a means for requesting the unregistration for simultaneous multiple contextEngineID values, e.g., all contextEngineID values, and may also provide a means for requesting simultaneous unregistration for multiple values of pduType.

2) If the contextEngineID and pduType combination has been registered, then the registration is deleted.

If no such registration exists, then the request is ignored.

5. Definitions

5.1. Definitions for SNMP Message Processing and Dispatching

```
SNMP-MPD-MIB DEFINITIONS ::= BEGIN

IMPORTS
    MODULE-COMPLIANCE, OBJECT-GROUP         FROM SNMPv2-CONF
    MODULE-IDENTITY, OBJECT-TYPE,
    snmpModules, Counter32                  FROM SNMPv2-SMI;

snmpMPDMIB MODULE-IDENTITY
    LAST-UPDATED "200210140000Z"
    ORGANIZATION "SNMPv3 Working Group"
    CONTACT-INFO "WG-EMail:   snmpv3@lists.tislabs.com
                  Subscribe:  snmpv3-request@lists.tislabs.com

                  Co-Chair:   Russ Mundy
                              Network Associates Laboratories
                  postal:     15204 Omega Drive, Suite 300
                              Rockville, MD 20850-4601
                              USA
```

```
                    EMail:      mundy@tislabs.com
                    phone:      +1 301-947-7107

                    Co-Chair &
                    Co-editor:  David Harrington
                                Enterasys Networks
                    postal:     35 Industrial Way
                                P. O. Box 5005
                                Rochester NH 03866-5005
                                USA
                    EMail:      dbh@enterasys.com
                    phone:      +1 603-337-2614

                    Co-editor:  Jeffrey Case
                                SNMP Research, Inc.
                    postal:     3001 Kimberlin Heights Road
                                Knoxville, TN 37920-9716
                                USA
                    EMail:      case@snmp.com
                    phone:      +1 423-573-1434

                    Co-editor:  Randy Presuhn
                                BMC Software, Inc.
                    postal:     2141 North First Street
                                San Jose, CA 95131
                                USA
                    EMail:      randy_presuhn@bmc.com
                    phone:      +1 408-546-1006

                    Co-editor:  Bert Wijnen
                                Lucent Technologies
                    postal:     Schagen 33
                                3461 GL Linschoten
                                Netherlands
                    EMail:      bwijnen@lucent.com
                    phone:      +31 348-680-485
                   "
        DESCRIPTION  "The MIB for Message Processing and Dispatching

                    Copyright (C) The Internet Society (2002). This
                    version of this MIB module is part of RFC 3412;
                    see the RFC itself for full legal notices.
                   "
        REVISION     "200210140000Z"            -- 14 October 2002
        DESCRIPTION  "Updated addresses, published as RFC 3412."
        REVISION     "199905041636Z"            -- 4 May 1999
        DESCRIPTION  "Updated addresses, published as RFC 2572."
```

```
        REVISION     "199709300000Z"            -- 30 September 1997
        DESCRIPTION  "Original version, published as RFC 2272."
        ::= { snmpModules 11 }

    -- Administrative assignments ****************************************

    snmpMPDAdmin           OBJECT IDENTIFIER ::= { snmpMPDMIB 1 }
    snmpMPDMIBObjects      OBJECT IDENTIFIER ::= { snmpMPDMIB 2 }
    snmpMPDMIBConformance  OBJECT IDENTIFIER ::= { snmpMPDMIB 3 }

    -- Statistics for SNMP Messages **************************************

    snmpMPDStats           OBJECT IDENTIFIER ::= { snmpMPDMIBObjects 1 }

    snmpUnknownSecurityModels OBJECT-TYPE
        SYNTAX       Counter32
        MAX-ACCESS   read-only
        STATUS       current
        DESCRIPTION "The total number of packets received by the SNMP
                     engine which were dropped because they referenced a
                     securityModel that was not known to or supported by
                     the SNMP engine.
                    "
        ::= { snmpMPDStats 1 }

    snmpInvalidMsgs OBJECT-TYPE
        SYNTAX       Counter32
        MAX-ACCESS   read-only
        STATUS       current
        DESCRIPTION "The total number of packets received by the SNMP
                     engine which were dropped because there were invalid
                     or inconsistent components in the SNMP message.
                    "
        ::= { snmpMPDStats 2 }

    snmpUnknownPDUHandlers OBJECT-TYPE
        SYNTAX       Counter32
        MAX-ACCESS   read-only
        STATUS       current
        DESCRIPTION "The total number of packets received by the SNMP
                     engine which were dropped because the PDU contained
                     in the packet could not be passed to an application
                     responsible for handling the pduType, e.g. no SNMP
                     application had registered for the proper
                     combination of the contextEngineID and the pduType.
                    "
        ::= { snmpMPDStats 3 }
```

```
   -- Conformance information ****************************************

   snmpMPDMIBCompliances OBJECT IDENTIFIER ::= {snmpMPDMIBConformance 1}
   snmpMPDMIBGroups      OBJECT IDENTIFIER ::= {snmpMPDMIBConformance 2}

   -- Compliance statements

   snmpMPDCompliance MODULE-COMPLIANCE
       STATUS       current
       DESCRIPTION "The compliance statement for SNMP entities which
                    implement the SNMP-MPD-MIB.
                   "
       MODULE    -- this module
           MANDATORY-GROUPS { snmpMPDGroup }
       ::= { snmpMPDMIBCompliances 1 }

   snmpMPDGroup OBJECT-GROUP
       OBJECTS {
                 snmpUnknownSecurityModels,
                 snmpInvalidMsgs,
                 snmpUnknownPDUHandlers
               }
       STATUS       current
       DESCRIPTION "A collection of objects providing for remote
                    monitoring of the SNMP Message Processing and
                    Dispatching process.
                   "
       ::= { snmpMPDMIBGroups 1 }

   END
```

6. The SNMPv3 Message Format

This section defines the SNMPv3 message format and the corresponding SNMP version 3 Message Processing Model (v3MP).

```
   SNMPv3MessageSyntax DEFINITIONS IMPLICIT TAGS ::= BEGIN

       SNMPv3Message ::= SEQUENCE {
           -- identify the layout of the SNMPv3Message
           -- this element is in same position as in SNMPv1
           -- and SNMPv2c, allowing recognition
           -- the value 3 is used for snmpv3
           msgVersion INTEGER ( 0 .. 2147483647 ),
           -- administrative parameters
           msgGlobalData HeaderData,
           -- security model-specific parameters
           -- format defined by Security Model
```

```
            msgSecurityParameters OCTET STRING,
            msgData  ScopedPduData
        }

        HeaderData ::= SEQUENCE {
            msgID      INTEGER (0..2147483647),
            msgMaxSize INTEGER (484..2147483647),

            msgFlags   OCTET STRING (SIZE(1)),
                       --  .... ...1   authFlag
                       --  .... ..1.   privFlag
                       --  .... .1..   reportableFlag
                       --              Please observe:
                       --  .... ..00   is OK, means noAuthNoPriv
                       --  .... ..01   is OK, means authNoPriv
                       --  .... ..10   reserved, MUST NOT be used.
                       --  .... ..11   is OK, means authPriv

            msgSecurityModel INTEGER (1..2147483647)
        }

        ScopedPduData ::= CHOICE {
            plaintext    ScopedPDU,
            encryptedPDU OCTET STRING  -- encrypted scopedPDU value
        }

        ScopedPDU ::= SEQUENCE {
            contextEngineID  OCTET STRING,
            contextName      OCTET STRING,
            data             ANY -- e.g., PDUs as defined in [RFC3416]
        }
    END
```

6.1. msgVersion

The msgVersion field is set to snmpv3(3) and identifies the message as an SNMP version 3 Message.

6.2. msgID

The msgID is used between two SNMP entities to coordinate request messages and responses, and by the v3MP to coordinate the processing of the message by different subsystem models within the architecture.

Values for msgID SHOULD be generated in a manner that avoids re-use of any outstanding values. Doing so provides protection against some replay attacks. One possible implementation strategy would be to use the low-order bits of snmpEngineBoots [RFC3411] as the high-order

portion of the msgID value and a monotonically increasing integer for the low-order portion of msgID.

Note that the request-id in a PDU may be used by SNMP applications to identify the PDU; the msgID is used by the engine to identify the message which carries a PDU. The engine needs to identify the message even if decryption of the PDU (and request-id) fails. No assumption should be made that the value of the msgID and the value of the request-id are equivalent.

The value of the msgID field for a response takes the value of the msgID field from the message to which it is a response. By use of the msgID value, an engine can distinguish the (potentially multiple) outstanding requests, and thereby correlate incoming responses with outstanding requests. In cases where an unreliable datagram service is used, the msgID also provides a simple means of identifying messages duplicated by the network. If a request is retransmitted, a new msgID value SHOULD be used for each retransmission.

6.3. msgMaxSize

The msgMaxSize field of the message conveys the maximum message size supported by the sender of the message, i.e., the maximum message size that the sender can accept when another SNMP engine sends an SNMP message (be it a response or any other message) to the sender of this message on the transport in use for this message.

When an SNMP message is being generated, the msgMaxSize is provided by the SNMP engine which generates the message. At the receiving SNMP engine, the msgMaxSize is used to determine the maximum message size the sender can accommodate.

6.4. msgFlags

The msgFlags field of the message contains several bit fields which control processing of the message.

The reportableFlag is a secondary aid in determining whether a Report PDU MUST be sent. It is only used in cases where the PDU portion of a message cannot be decoded, due to, for example, an incorrect encryption key. If the PDU can be decoded, the PDU type forms the basis for decisions on sending Report PDUs.

When the reportableFlag is used, if its value is one, a Report PDU MUST be returned to the sender under those conditions which can cause the generation of Report PDUs. Similarly, when the reportableFlag is used and its value is zero, then a Report PDU MUST NOT be sent. The reportableFlag MUST always be zero when the message contains a PDU

from the Unconfirmed Class, such as a Report PDU, a response-type PDU (such as a Response PDU), or an unacknowledged notification-type PDU (such as an SNMPv2-trap PDU). The reportableFlag MUST always be one for a PDU from the Confirmed Class, including request-type PDUs (such as a Get PDU) and acknowledged notification-type PDUs (such as an Inform PDU).

If the reportableFlag is set to one for a message containing a PDU from the Unconfirmed Class, such as a Report PDU, a response-type PDU (such as a Response PDU), or an unacknowledged notification-type PDU (such as an SNMPv2-trap PDU), then the receiver of that message MUST process it as though the reportableFlag had been set to zero.

If the reportableFlag is set to zero for a message containing a request-type PDU (such as a Get PDU) or an acknowledged notification-type PDU (such as an Inform PDU), then the receiver of that message MUST process it as though the reportableFlag had been set to one.

Report PDUs are generated directly by the SNMPv3 Message Processing Model, and support engine-to-engine communications, but may be passed to applications for processing.

An SNMP engine that receives a reportPDU may use it to determine what kind of problem was detected by the remote SNMP engine. It can do so based on the error counter included as the first (and only) varBind of the reportPDU. Based on the detected error, the SNMP engine may try to send a corrected SNMP message. If that is not possible, it may pass an indication of the error to the application on whose behalf the failed SNMP request was issued.

The authFlag and privFlag portions of the msgFlags field are set by the sender to indicate the securityLevel that was applied to the message before it was sent on the wire. The receiver of the message MUST apply the same securityLevel when the message is received and the contents are being processed.

There are three securityLevels, namely noAuthNoPriv, which is less than authNoPriv, which is in turn less than authPriv. See the SNMP architecture document [RFC3411] for details about the securityLevel.

a) authFlag

   If the authFlag is set to one, then the securityModel used by the SNMP engine which sent the message MUST identify the securityName on whose behalf the SNMP message was generated and MUST provide, in a securityModel-specific manner, sufficient data for the receiver of the message to be able to authenticate that

identification. In general, this authentication will allow the receiver to determine with reasonable certainty that the message was:

- sent on behalf of the principal associated with the securityName,
- was not redirected,
- was not modified in transit, and
- was not replayed.

If the authFlag is zero, then the securityModel used by the SNMP engine which sent the message MUST identify the securityName on whose behalf the SNMP message was generated but it does not need to provide sufficient data for the receiver of the message to authenticate the identification, as there is no need to authenticate the message in this case.

b) privFlag

If the privFlag is set, then the securityModel used by the SNMP engine which sent the message MUST also protect the scopedPDU in an SNMP message from disclosure, i.e., it MUST encrypt/decrypt the scopedPDU. If the privFlag is zero, then the securityModel in use does not need to protect the data from disclosure.

It is an explicit requirement of the SNMP architecture that if privacy is selected, then authentication is also required. That means that if the privFlag is set, then the authFlag MUST also be set to one.

The combination of the authFlag and the privFlag comprises a Level of Security as follows:

```
authFlag zero, privFlag zero -> securityLevel is noAuthNoPriv
authFlag zero, privFlag one  -> invalid combination, see below
authFlag one,  privFlag zero -> securityLevel is authNoPriv
authFlag one,  privFlag one  -> securityLevel is authPriv
```

The elements of procedure (see below) describe the action to be taken when the invalid combination of authFlag equal to zero and privFlag equal to one is encountered.

The remaining bits in msgFlags are reserved, and MUST be set to zero when sending a message and SHOULD be ignored when receiving a message.

6.5. msgSecurityModel

The v3MP supports the concurrent existence of multiple Security Models to provide security services for SNMPv3 messages. The msgSecurityModel field in an SNMPv3 Message identifies which Security Model was used by the sender to generate the message and therefore which securityModel MUST be used by the receiver to perform security processing for the message. The mapping to the appropriate securityModel implementation within an SNMP engine is accomplished in an implementation-dependent manner.

6.6. msgSecurityParameters

The msgSecurityParameters field of the SNMPv3 Message is used for communication between the Security Model modules in the sending and receiving SNMP engines. The data in the msgSecurityParameters field is used exclusively by the Security Model, and the contents and format of the data is defined by the Security Model. This OCTET STRING is not interpreted by the v3MP, but is passed to the local implementation of the Security Model indicated by the msgSecurityModel field in the message.

6.7. scopedPduData

The scopedPduData field represents either the plain text scopedPDU if the privFlag in the msgFlags is zero, or it represents an encryptedPDU (encoded as an OCTET STRING) which MUST be decrypted by the securityModel in use to produce a plaintext scopedPDU.

6.8. scopedPDU

The scopedPDU contains information to identify an administratively unique context and a PDU. The object identifiers in the PDU refer to managed objects which are (expected to be) accessible within the specified context.

6.8.1. contextEngineID

The contextEngineID in the SNMPv3 message uniquely identifies, within an administrative domain, an SNMP entity that may realize an instance of a context with a particular contextName.

For incoming messages, the contextEngineID is used in conjunction with the pduType to determine to which application the scopedPDU will be sent for processing.

For outgoing messages, the v3MP sets the contextEngineID to the value provided by the application in the request for a message to be sent.

6.8.2. contextName

The contextName field in an SNMPv3 message, in conjunction with the contextEngineID field, identifies the particular context associated with the management information contained in the PDU portion of the message. The contextName is unique within the SNMP entity specified by the contextEngineID, which may realize the managed objects referenced within the PDU. An application which originates a message provides the value for the contextName field and this value may be used during processing by an application at the receiving SNMP Engine.

6.8.3. data

The data field of the SNMPv3 Message contains the PDU. Among other things, the PDU contains the PDU type that is used by the v3MP to determine the type of the incoming SNMP message. The v3MP specifies that the PDU MUST be one of those specified in [RFC3416].

7. Elements of Procedure for v3MP

This section describes the procedures followed by an SNMP engine when generating and processing SNMP messages according to the SNMPv3 Message Processing Model.

Please note, that for the sake of clarity and to prevent the text from being even longer and more complicated, some details were omitted from the steps below.

a) Some steps specify that when some error conditions are encountered when processing a received message, a message containing a Report PDU is generated and the received message is discarded without further processing. However, a Report-PDU MUST NOT be generated unless the PDU causing generation of the Report PDU can be determined to be a member of the Confirmed Class, or the reportableFlag is set to one and the PDU class cannot be determined.

b) The elements of procedure do not always explicitly indicate when state information needs to be released. The general rule is that if state information is available when a message is to be "discarded without further processing", then the state information should also be released at that same time.

7.1. Prepare an Outgoing SNMP Message

This section describes the procedure followed to prepare an SNMPv3 message from the data elements passed by the Message Dispatcher.

1) The Message Dispatcher may request that an SNMPv3 message containing a Read Class, Write Class, or Notification Class PDU be prepared for sending.

   a) It makes such a request according to the abstract service primitive:

```
statusInformation =           -- success or errorIndication
  prepareOutgoingMessage(
  IN   transportDomain        -- requested transport domain
  IN   transportAddress       -- requested destination address
  IN   messageProcessingModel -- typically, SNMP version
  IN   securityModel          -- Security Model to use
  IN   securityName           -- on behalf of this principal
  IN   securityLevel          -- Level of Security requested
  IN   contextEngineID        -- data from/at this entity
  IN   contextName            -- data from/in this context
  IN   pduVersion             -- version of the PDU *
  IN   PDU                    -- SNMP Protocol Data Unit
  IN   expectResponse         -- TRUE or FALSE *
  IN   sendPduHandle          -- the handle for matching
                              -- incoming responses
  OUT  destTransportDomain    -- destination transport domain
  OUT  destTransportAddress   -- destination transport address
  OUT  outgoingMessage        -- the message to send
  OUT  outgoingMessageLength  -- the length of the message
  )
```

   * The SNMPv3 Message Processing Model does not use the values of expectResponse or pduVersion.

   b) A unique msgID is generated. The number used for msgID should not have been used recently, and MUST NOT be the same as was used for any outstanding request.

2) The Message Dispatcher may request that an SNMPv3 message containing a Response Class or Internal Class PDU be prepared for sending.

a) It makes such a request according to the abstract service primitive:

```
result =                        -- SUCCESS or FAILURE
prepareResponseMessage(
 IN   messageProcessingModel    -- typically, SNMP version
 IN   securityModel             -- same as on incoming request
 IN   securityName              -- same as on incoming request
 IN   securityLevel             -- same as on incoming request
 IN   contextEngineID           -- data from/at this SNMP entity
 IN   contextName               -- data from/in this context
 IN   pduVersion                -- version of the PDU
 IN   PDU                       -- SNMP Protocol Data Unit
 IN   maxSizeResponseScopedPDU  -- maximum size sender can
                                -- accept
 IN   stateReference            -- reference to state
                                -- information presented with
                                -- the request
 IN   statusInformation         -- success or errorIndication
                                -- error counter OID and value
                                -- when errorIndication
 OUT  destTransportDomain       -- destination transport domain
 OUT  destTransportAddress      -- destination transport address
 OUT  outgoingMessage           -- the message to send
 OUT  outgoingMessageLength     -- the length of the message
 )
```

b) The cached information for the original request is retrieved via the stateReference, including:

- msgID,
- contextEngineID,
- contextName,
- securityModel,
- securityName,
- securityLevel,
- securityStateReference,
- reportableFlag,
- transportDomain, and
- transportAddress.

The SNMPv3 Message Processing Model does not allow cached data to be overridden, except by error indications as detailed in (3) below.

3) If statusInformation contains values for an OID/value combination (potentially also containing a securityLevel value, contextEngineID value, or contextName value), then:

   a) If a PDU is provided, it is the PDU from the original request. If possible, extract the request-id and pduType.

   b) If the pduType is determined to not be a member of the Confirmed Class, or if the reportableFlag is zero and the pduType cannot be determined, then the original message is discarded, and no further processing is done. A result of FAILURE is returned. SNMPv3 Message Processing is complete.

   c) A Report PDU is prepared:

      1) the varBindList is set to contain the OID and value from the statusInformation.

      2) error-status is set to 0.

      3) error-index is set to 0.

      4) request-id is set to the value extracted in step b). Otherwise, request-id is set to 0.

   d) The errorIndication in statusInformation may be accompanied by a securityLevel value, a contextEngineID value, or a contextName value.

      1) If statusInformation contains a value for securityLevel, then securityLevel is set to that value, otherwise it is set to noAuthNoPriv.

      2) If statusInformation contains a value for contextEngineID, then contextEngineID is set to that value, otherwise it is set to the value of this entity’s snmpEngineID.

      3) If statusInformation contains a value for contextName, then contextName is set to that value, otherwise it is set to the default context of "" (zero-length string).

   e) PDU is set to refer to the new Report-PDU. The old PDU is discarded.

   f) Processing continues with step 6) below.

4) If the contextEngineID is not yet determined, then the contextEngineID is determined, in an implementation-dependent manner, possibly using the transportDomain and transportAddress.

5) If the contextName is not yet determined, the contextName is set to the default context.

6) A scopedPDU is prepared from the contextEngineID, contextName, and PDU.

7) msgGlobalData is constructed as follows:

   a) The msgVersion field is set to snmpv3(3).

   b) msgID is set as determined in step 1 or 2 above.

   c) msgMaxSize is set to an implementation-dependent value.

   d) msgFlags are set as follows:

      - If securityLevel specifies noAuthNoPriv, then authFlag and privFlag are both set to zero.

      - If securityLevel specifies authNoPriv, then authFlag is set to one and privFlag is set to zero.

      - If securityLevel specifies authPriv, then authFlag is set to one and privFlag is set to one.

      - If the PDU is from the Unconfirmed Class, then the reportableFlag is set to zero.

      - If the PDU is from the Confirmed Class then the reportableFlag is set to one.

      - All other msgFlags bits are set to zero.

   e) msgSecurityModel is set to the value of securityModel.

8) If the PDU is from the Response Class or the Internal Class, then:

   a) The specified Security Model is called to generate the message according to the primitive:

```
statusInformation =
  generateResponseMsg(
  IN   messageProcessingModel -- SNMPv3 Message Processing
                              -- Model
  IN   globalData             -- msgGlobalData from step 7
  IN   maxMessageSize         -- from msgMaxSize (step 7c)
  IN   securityModel          -- as determined in step 7e
  IN   securityEngineID       -- the value of snmpEngineID
  IN   securityName           -- on behalf of this principal
  IN   securityLevel          -- for the outgoing message
  IN   scopedPDU              -- as prepared in step 6)
  IN   securityStateReference -- as determined in step 2
  OUT  securityParameters     -- filled in by Security Module
  OUT  wholeMsg               -- complete generated message
  OUT  wholeMsgLength         -- length of generated message
  )
```

      If, upon return from the Security Model, the statusInformation includes an errorIndication, then any cached information about the outstanding request message is discarded, and an errorIndication is returned, so it can be returned to the calling application. SNMPv3 Message Processing is complete.

   b) A SUCCESS result is returned. SNMPv3 Message Processing is complete.

9) If the PDU is from the Confirmed Class or the Notification Class, then:

   a) If the PDU is from the Unconfirmed Class, then securityEngineID is set to the value of this entity's snmpEngineID.

      Otherwise, the snmpEngineID of the target entity is determined, in an implementation-dependent manner, possibly using transportDomain and transportAddress. The value of the securityEngineID is set to the value of the target entity's snmpEngineID.

b) The specified Security Model is called to generate the message according to the primitive:

```
statusInformation =
 generateRequestMsg(
 IN  messageProcessingModel -- SNMPv3 Message Processing Model
 IN  globalData             -- msgGlobalData, from step 7
 IN  maxMessageSize         -- from msgMaxSize in step 7 c)
 IN  securityModel          -- as provided by caller
 IN  securityEngineID       -- authoritative SNMP entity
                            -- from step 9 a)
 IN  securityName           -- as provided by caller
 IN  securityLevel          -- as provided by caller
 IN  scopedPDU              -- as prepared in step 6
 OUT securityParameters     -- filled in by Security Module
 OUT wholeMsg               -- complete generated message
 OUT wholeMsgLength         -- length of the generated message
 )
```

If, upon return from the Security Model, the statusInformation includes an errorIndication, then the message is discarded, and the errorIndication is returned, so it can be returned to the calling application, and no further processing is done. SNMPv3 Message Processing is complete.

c) If the PDU is from the Confirmed Class, information about the outgoing message is cached, and an implementation-specific stateReference is created. Information to be cached includes the values of:

- sendPduHandle
- msgID
- snmpEngineID
- securityModel
- securityName
- securityLevel
- contextEngineID
- contextName

d) A SUCCESS result is returned. SNMPv3 Message Processing is complete.

7.2. Prepare Data Elements from an Incoming SNMP Message

This section describes the procedure followed to extract data from an SNMPv3 message, and to prepare the data elements required for further processing of the message by the Message Dispatcher.

1) The message is passed in from the Message Dispatcher according to the abstract service primitive:

```
result =                        -- SUCCESS or errorIndication
  prepareDataElements(
  IN  transportDomain           -- origin transport domain
  IN  transportAddress          -- origin transport address
  IN  wholeMsg                  -- as received from the network
  IN  wholeMsgLength            -- as received from the network
  OUT messageProcessingModel    -- typically, SNMP version
  OUT securityModel             -- Security Model to use
  OUT securityName              -- on behalf of this principal
  OUT securityLevel             -- Level of Security requested
  OUT contextEngineID           -- data from/at this entity
  OUT contextName               -- data from/in this context
  OUT pduVersion                -- version of the PDU
  OUT PDU                       -- SNMP Protocol Data Unit
  OUT pduType                   -- SNMP PDU type
  OUT sendPduHandle             -- handle for matched request
  OUT maxSizeResponseScopedPDU -- maximum size sender can accept
  OUT statusInformation         -- success or errorIndication
                                -- error counter OID and value
                                -- when errorIndication
  OUT stateReference            -- reference to state information
                                -- to be used for a possible
  )                             -- Response
```

2) If the received message is not the serialization (according to the conventions of [RFC3417]) of an SNMPv3Message value, then the snmpInASNParseErrs counter [RFC3418] is incremented, the message is discarded without further processing, and a FAILURE result is returned. SNMPv3 Message Processing is complete.

3) The values for msgVersion, msgID, msgMaxSize, msgFlags, msgSecurityModel, msgSecurityParameters, and msgData are extracted from the message.

4) If the value of the msgSecurityModel component does not match a supported securityModel, then the snmpUnknownSecurityModels counter is incremented, the message is discarded without further processing, and a FAILURE result is returned. SNMPv3 Message Processing is complete.

5) The securityLevel is determined from the authFlag and the privFlag bits of the msgFlags component as follows:

   a) If the authFlag is not set and the privFlag is not set, then securityLevel is set to noAuthNoPriv.

   b) If the authFlag is set and the privFlag is not set, then securityLevel is set to authNoPriv.

   c) If the authFlag is set and the privFlag is set, then securityLevel is set to authPriv.

   d) If the authFlag is not set and privFlag is set, then the snmpInvalidMsgs counter is incremented, the message is discarded without further processing, and a FAILURE result is returned. SNMPv3 Message Processing is complete.

   e) Any other bits in the msgFlags are ignored.

6) The security module implementing the Security Model as specified by the securityModel component is called for authentication and privacy services. This is done according to the abstract service primitive:

```
statusInformation =            -- errorIndication or success
                               -- error counter OID and
                               -- value if error
  processIncomingMsg(
  IN  messageProcessingModel   -- SNMPv3 Message Processing Model
  IN  maxMessageSize           -- of the sending SNMP entity
  IN  securityParameters       -- for the received message
  IN  securityModel            -- for the received message
  IN  securityLevel            -- Level of Security
  IN  wholeMsg                 -- as received on the wire
  IN  wholeMsgLength           -- length as received on the wire
  OUT securityEngineID         -- authoritative SNMP entity
  OUT securityName             -- identification of the principal
  OUT scopedPDU,               -- message (plaintext) payload
  OUT maxSizeResponseScopedPDU -- maximum size sender can accept
  OUT securityStateReference   -- reference to security state
  )                            -- information, needed for
                               -- response
```

   If an errorIndication is returned by the security module, then:

   a) If statusInformation contains values for an OID/value pair, then generation of a Report PDU is attempted (see step 3 in section 7.1).

1) If the scopedPDU has been returned from processIncomingMsg, then determine contextEngineID, contextName, and PDU.

2) Information about the message is cached and a stateReference is created (implementation-specific). Information to be cached includes the values of:

```
msgVersion,
msgID,
securityLevel,
msgFlags,
msgMaxSize,
securityModel,
maxSizeResponseScopedPDU,
securityStateReference
```

3) Request that a Report-PDU be prepared and sent, according to the abstract service primitive:

```
result =                      -- SUCCESS or FAILURE
returnResponsePdu(
IN  messageProcessingModel    -- SNMPv3(3)
IN  securityModel             -- same as on incoming request
IN  securityName              -- from processIncomingMsg
IN  securityLevel             -- same as on incoming request
IN  contextEngineID           -- from step 6 a) 1)
IN  contextName               -- from step 6 a) 1)
IN  pduVersion                -- SNMPv2-PDU
IN  PDU                       -- from step 6 a) 1)
IN  maxSizeResponseScopedPDU  -- from processIncomingMsg
IN  stateReference            -- from step 6 a) 2)
IN  statusInformation         -- from processIncomingMsg
)
```

b) The incoming message is discarded without further processing, and a FAILURE result is returned. SNMPv3 Message Processing is complete.

7) The scopedPDU is parsed to extract the contextEngineID, the contextName and the PDU. If any parse error occurs, then the snmpInASNParseErrs counter [RFC3418] is incremented, the security state information is discarded, the message is discarded without further processing, and a FAILURE result is returned. SNMPv3 Message Processing is complete. Treating an unknown PDU type is treated as a parse error is an implementation option.

8) The pduVersion is determined in an implementation-dependent manner. For SNMPv3, the pduVersion would be an SNMPv2-PDU.

9) The pduType is determined, in an implementation-dependent manner. For [RFC3416], the pduTypes include:

- GetRequest-PDU,
- GetNextRequest-PDU,
- GetBulkRequest-PDU,
- SetRequest-PDU,
- InformRequest-PDU,
- SNMPv2-Trap-PDU,
- Response-PDU,
- Report-PDU.

10) If the pduType is from the Response Class or the Internal Class, then:

a) The value of the msgID component is used to find the cached information for a corresponding outstanding Request message. If no such outstanding Request message is found, then the security state information is discarded, the message is discarded without further processing, and a FAILURE result is returned. SNMPv3 Message Processing is complete.

b) sendPduHandle is retrieved from the cached information.

Otherwise, sendPduHandle is set to <none>, an implementation defined value.

11) If the pduType is from the Internal Class, then:

a) statusInformation is created using the contents of the Report-PDU, in an implementation-dependent manner. This statusInformation will be forwarded to the application associated with the sendPduHandle.

b) The cached data for the outstanding message, referred to by stateReference, is retrieved. If the securityModel or securityLevel values differ from the cached ones, it is important to recognize that Internal Class PDUs delivered at the security level of noAuthNoPriv open a window of opportunity for spoofing or replay attacks. If the receiver of such messages is aware of these risks, the use of such unauthenticated messages is acceptable and may provide a useful function for discovering engine IDs or for detecting misconfiguration at remote nodes.

When the securityModel or securityLevel values differ from the cached ones, an implementation may retain the cached information about the outstanding Request message, in anticipation of the possibility that the Internal Class PDU received might be illegitimate. Otherwise, any cached information about the outstanding Request message is discarded.

c) The security state information for this incoming message is discarded.

d) stateReference is set to <none>.

e) A SUCCESS result is returned. SNMPv3 Message Processing is complete.

12) If the pduType is from the Response Class, then:

a) The cached data for the outstanding request, referred to by stateReference, is retrieved, including:

- snmpEngineID
- securityModel
- securityName
- securityLevel
- contextEngineID
- contextName

b) If the values extracted from the incoming message differ from the cached data, then any cached information about the outstanding Request message is discarded, the incoming message is discarded without further processing, and a FAILURE result is returned. SNMPv3 Message Processing is complete.

When the securityModel or securityLevel values differ from the cached ones, an implementation may retain the cached information about the outstanding Request message, in anticipation of the possibility that the Response Class PDU received might be illegitimate.

c) Otherwise, any cached information about the outstanding Request message is discarded, and the stateReference is set to <none>.

d) A SUCCESS result is returned. SNMPv3 Message Processing is complete.

13) If the pduType is from the Confirmed Class, then:

a) If the value of securityEngineID is not equal to the value of snmpEngineID, then the security state information is discarded, any cached information about this message is discarded, the incoming message is discarded without further processing, and a FAILURE result is returned. SNMPv3 Message Processing is complete.

b) Information about the message is cached and a stateReference is created (implementation-specific). Information to be cached includes the values of:

```
msgVersion,
msgID,
securityLevel,
msgFlags,
msgMaxSize,
securityModel,
maxSizeResponseScopedPDU,
securityStateReference
```

c) A SUCCESS result is returned. SNMPv3 Message Processing is complete.

14) If the pduType is from the Unconfirmed Class, then a SUCCESS result is returned. SNMPv3 Message Processing is complete.

8. Intellectual Property

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-related documentation can be found in BCP-11. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to 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.

9. Acknowledgements

This document is the result of the efforts of the SNMPv3 Working Group. Some special thanks are in order to the following SNMPv3 WG members:

Harald Tveit Alvestrand (Maxware)
Dave Battle (SNMP Research, Inc.)
Alan Beard (Disney Worldwide Services)
Paul Berrevoets (SWI Systemware/Halcyon Inc.)
Martin Bjorklund (Ericsson)
Uri Blumenthal (IBM T. J. Watson Research Center)
Jeff Case (SNMP Research, Inc.)
John Curran (BBN)
Mike Daniele (Compaq Computer Corporation)
T. Max Devlin (Eltrax Systems)
John Flick (Hewlett Packard)
Rob Frye (MCI)
Wes Hardaker (U.C.Davis, Information Technology - D.C.A.S.)
David Harrington (Cabletron Systems Inc.)
Lauren Heintz (BMC Software, Inc.)
N.C. Hien (IBM T. J. Watson Research Center)
Michael Kirkham (InterWorking Labs, Inc.)
Dave Levi (SNMP Research, Inc.)
Louis A Mamakos (UUNET Technologies Inc.)
Joe Marzot (Nortel Networks)
Paul Meyer (Secure Computing Corporation)
Keith McCloghrie (Cisco Systems)
Bob Moore (IBM)
Russ Mundy (TIS Labs at Network Associates)
Bob Natale (ACE*COMM Corporation)
Mike O'Dell (UUNET Technologies Inc.)
Dave Perkins (DeskTalk)
Peter Polkinghorne (Brunel University)
Randy Presuhn (BMC Software, Inc.)
David Reeder (TIS Labs at Network Associates)
David Reid (SNMP Research, Inc.)
Aleksey Romanov (Quality Quorum)
Shawn Routhier (Epilogue)
Juergen Schoenwaelder (TU Braunschweig)
Bob Stewart (Cisco Systems)
Mike Thatcher (Independent Consultant)
Bert Wijnen (IBM T. J. Watson Research Center)

The document is based on recommendations of the IETF Security and Administrative Framework Evolution for SNMP Advisory Team. Members of that Advisory Team were:

David Harrington (Cabletron Systems Inc.)
Jeff Johnson (Cisco Systems)
David Levi (SNMP Research Inc.)
John Linn (Openvision)
Russ Mundy (Trusted Information Systems) chair
Shawn Routhier (Epilogue)
Glenn Waters (Nortel)
Bert Wijnen (IBM T. J. Watson Research Center)

As recommended by the Advisory Team and the SNMPv3 Working Group Charter, the design incorporates as much as practical from previous RFCs and drafts. As a result, special thanks are due to the authors of previous designs known as SNMPv2u and SNMPv2*:

Jeff Case (SNMP Research, Inc.)
David Harrington (Cabletron Systems Inc.)
David Levi (SNMP Research, Inc.)
Keith McCloghrie (Cisco Systems)
Brian O'Keefe (Hewlett Packard)
Marshall T. Rose (Dover Beach Consulting)
Jon Saperia (BGS Systems Inc.)
Steve Waldbusser (International Network Services)
Glenn W. Waters (Bell-Northern Research Ltd.)

10. Security Considerations

The Dispatcher coordinates the processing of messages to provide a level of security for management messages and to direct the SNMP PDUs to the proper SNMP application(s).

A Message Processing Model, and in particular the v3MP defined in this document, interacts as part of the Message Processing with Security Models in the Security Subsystem via the abstract service interface primitives defined in [RFC3411] and elaborated above.

The level of security actually provided is primarily determined by the specific Security Model implementation(s) and the specific SNMP application implementation(s) incorporated into this framework. Applications have access to data which is not secured. Applications should take reasonable steps to protect the data from disclosure, and when they send data across the network, they should obey the securityLevel and call upon the services of an Access Control Model as they apply access control.

The values for the msgID element used in communication between SNMP entities MUST be chosen to avoid replay attacks. The values do not need to be unpredictable; it is sufficient that they not repeat.

When exchanges are carried out over an insecure network, there is an open opportunity for a third party to spoof or replay messages when any message of an exchange is given at the security level of noAuthNoPriv. For most exchanges, all messages exist at the same security level. In the case where the final message is an Internal Class PDU, this message may be delivered at a level of noAuthNoPriv or authNoPriv, independent of the security level of the preceding messages. Internal Class PDUs delivered at the level of authNoPriv are not considered to pose a security hazard. Internal Class PDUs delivered at the security level of noAuthNoPriv open a window of opportunity for spoofing or replay attacks. If the receiver of such messages is aware of these risks, the use of such unauthenticated messages is acceptable and may provide a useful function for discovering engine IDs or for detecting misconfiguration at remote nodes.

This document also contains a MIB definition module. None of the objects defined is writable, and the information they represent is not deemed to be particularly sensitive. However, if they are deemed sensitive in a particular environment, access to them should be restricted through the use of appropriately configured Security and Access Control models.

11. References

11.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2578] McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M. and S. Waldbusser, "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, April 1999.

[RFC2580] McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M. and S. Waldbusser, "Conformance Statements for SMIv2", STD 58, RFC 2580, April 1999.

[RFC3411] Harrington, D., Presuhn, R. and B. Wijnen, "An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks", STD 62, RFC 3411, December 2002.

[RFC3413] Levi, D., Meyer, P. and B. Stewart, "Simple Network Management Protocol (SNMP) Applications", STD 62, RFC 3413, December 2002.

[RFC3414] Blumenthal, U. and B. Wijnen, "The User-Based Security Model (USM) for Version 3 of the Simple Network Management Protocol (SNMPv3)", STD 62, RFC 3414, December 2002.

[RFC3415] 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.

[RFC3416] Presuhn, R., Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3416, December 2002.

[RFC3417] Presuhn, R., Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Transport Mappings for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3417, December 2002.

[RFC3418] Presuhn, R., Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Management Information Base (MIB) for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3418, December 2002.

11.2. Informative References

[RFC1901] Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Introduction to Community-based SNMPv2", RFC 1901, January 1996.

[RFC2028] Hovey, R. and S. Bradner, "The Organizations Involved in the IETF Standards Process", BCP 11, RFC 2028, October 1996.

[RFC2576] Frye, R., Levi, D., Routhier, S. and B. Wijnen, "Coexistence between Version 1, Version 2, and Version 3 of the Internet-Standard Network Management Framework", RFC 2576, March 2000.

[RFC3410] Case, J., Mundy, R., Partain, D. and B. Stewart, "Introduction and Applicability Statements for Internet-Standard Management Framework", RFC 3410, December 2002.

12. Editors' Addresses

Jeffrey Case
SNMP Research, Inc.
3001 Kimberlin Heights Road
Knoxville, TN 37920-9716
USA

Phone: +1 423-573-1434
EMail: case@snmp.com

David Harrington
Enterasys Networks
35 Industrial Way
Post Office Box 5005
Rochester, NH 03866-5005
USA

Phone: +1 603-337-2614
EMail: dbh@enterasys.com

Randy Presuhn
BMC Software, Inc.
2141 North First Street
San Jose, CA 95131
USA

Phone: +1 408-546-1006
EMail: randy_presuhn@bmc.com

Bert Wijnen
Lucent Technologies
Schagen 33
3461 GL Linschoten
Netherlands

Phone: +31 348-680-485
EMail: bwijnen@lucent.com

13. Full Copyright Statement

Acknowledgement

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

Network Working Group
Request for Comments: 3413
STD: 62
Obsoletes: 2573
Category: Standards Track

D. Levi
Nortel Networks
P. Meyer
Secure Computing Corporation
B. Stewart
Retired

December 2002

Simple Network Management Protocol (SNMP) Applications

Status of this Memo

Abstract

This document describes five types of Simple Network Management Protocol (SNMP) applications which make use of an SNMP engine as described in STD 62, RFC 3411. The types of application described are Command Generators, Command Responders, Notification Originators, Notification Receivers, and Proxy Forwarders.

This document also defines Management Information Base (MIB) modules for specifying targets of management operations, for notification filtering, and for proxy forwarding. This document obsoletes RFC 2573.

Table of Contents

1. Overview

This document describes five types of SNMP applications:

- Applications which initiate SNMP Read-Class, and/or Write-Class requests, called 'command generators.'

- Applications which respond to SNMP Read-Class, and/or Write-Class requests, called 'command responders.'

- Applications which generate SNMP Notification-Class PDUs, called 'notification originators.'

- Applications which receive SNMP Notification-Class PDUs, called 'notification receivers.'

- Applications which forward SNMP messages, called 'proxy forwarders.'

Note that there are no restrictions on which types of applications may be associated with a particular SNMP engine. For example, a single SNMP engine may, in fact, be associated with both command generator and command responder applications.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

1.1. Command Generator Applications

A command generator application initiates SNMP Read-Class and/or Write-Class requests, and processes responses to requests which it generated.

1.2. Command Responder Applications

A command responder application receives SNMP Read-Class and/or Write-Class requests destined for the local system as indicated by the fact that the contextEngineID in the received request is equal to that of the local engine through which the request was received. The command responder application will perform the appropriate protocol operation, using access control, and will generate a response message to be sent to the request's originator.

1.3. Notification Originator Applications

A notification originator application conceptually monitors a system for particular events or conditions, and generates Notification-Class messages based on these events or conditions. A notification originator must have a mechanism for determining where to send messages, and what SNMP version and security parameters to use when sending messages. A mechanism and MIB module for this purpose is provided in this document. Note that Notification-Class PDUs generated by a notification originator may be either Confirmed-Class or Unconfirmed-Class PDU types.

1.4. Notification Receiver Applications

A notification receiver application listens for notification messages, and generates response messages when a message containing a Confirmed-Class PDU is received.

1.5. Proxy Forwarder Applications

A proxy forwarder application forwards SNMP messages. Note that implementation of a proxy forwarder application is optional. The sections describing proxy (3.5, 4.3, and 7) may be skipped for implementations that do not include a proxy forwarder application.

The term "proxy" has historically been used very loosely, with multiple different meanings. These different meanings include (among others):

(1) the forwarding of SNMP requests to other SNMP entities without regard for what managed object types are being accessed; for example, in order to forward an SNMP request from one transport domain to another, or to translate SNMP requests of one version into SNMP requests of another version;

(2) the translation of SNMP requests into operations of some non-SNMP management protocol; and

(3) support for aggregated managed objects where the value of one managed object instance depends upon the values of multiple other (remote) items of management information.

Each of these scenarios can be advantageous; for example, support for aggregation of management information can significantly reduce the bandwidth requirements of large-scale management activities.

However, using a single term to cover multiple different scenarios causes confusion.

To avoid such confusion, this document uses the term "proxy" with a much more tightly defined meaning. The term "proxy" is used in this document to refer to a proxy forwarder application which forwards either SNMP messages without regard for what managed objects are contained within those messages. This definition is most closely related to the first definition above. Note, however, that in the SNMP architecture [RFC3411], a proxy forwarder is actually an application, and need not be associated with what is traditionally thought of as an SNMP agent.

Specifically, the distinction between a traditional SNMP agent and a proxy forwarder application is simple:

- a proxy forwarder application forwards SNMP messages to other SNMP engines according to the context, and irrespective of the specific managed object types being accessed, and forwards the response to such previously forwarded messages back to the SNMP engine from which the original message was received;

- in contrast, the command responder application that is part of what is traditionally thought of as an SNMP agent, and which processes SNMP requests according to the (names of the) individual managed object types and instances being accessed, is NOT a proxy forwarder application from the perspective of this document.

Thus, when a proxy forwarder application forwards a request or notification for a particular contextEngineID / contextName pair, not only is the information on how to forward the request specifically associated with that context, but the proxy forwarder application has no need of a detailed definition of a MIB view (since the proxy forwarder application forwards the request irrespective of the managed object types).

In contrast, a command responder application must have the detailed definition of the MIB view, and even if it needs to issue requests to other entities, via SNMP or otherwise, that need is dependent on the individual managed object instances being accessed (i.e., not only on the context).

Note that it is a design goal of a proxy forwarder application to act as an intermediary between the endpoints of a transaction. In particular, when forwarding Confirmed Notification-Class messages, the associated response is forwarded when it is received from the target to which the Notification-Class message was forwarded, rather than generating a response immediately when the Notification-Class message is received.

2. Management Targets

Some types of applications (notification generators and proxy forwarders in particular) require a mechanism for determining where and how to send generated messages. This document provides a mechanism and MIB module for this purpose. The set of information that describes where and how to send a message is called a 'Management Target', and consists of two kinds of information:

- Destination information, consisting of a transport domain and a transport address. This is also termed a transport endpoint.

- SNMP parameters, consisting of message processing model, security model, security level, and security name information.

The SNMP-TARGET-MIB module described later in this document contains one table for each of these types of information. There can be a many-to-many relationship in the MIB between these two types of information. That is, there may be multiple transport endpoints associated with a particular set of SNMP parameters, or a particular transport endpoint may be associated with several sets of SNMP parameters.

3. Elements Of Procedure

The following sections describe the procedures followed by each type of application when generating messages for transmission or when processing received messages. Applications communicate with the Dispatcher using the abstract service interfaces defined in [RFC3411].

3.1. Command Generator Applications

A command generator initiates an SNMP request by calling the Dispatcher using the following abstract service interface:

```
   statusInformation =              -- sendPduHandle if success
                                    -- errorIndication if failure
     sendPdu(
     IN   transportDomain           -- transport domain to be used
     IN   transportAddress          -- destination network address
     IN   messageProcessingModel    -- typically, SNMP version
     IN   securityModel             -- Security Model to use
     IN   securityName              -- on behalf of this principal
     IN   securityLevel             -- Level of Security requested
     IN   contextEngineID           -- data from/at this entity
     IN   contextName               -- data from/in this context
     IN   pduVersion                -- the version of the PDU
     IN   PDU                       -- SNMP Protocol Data Unit
     IN   expectResponse            -- TRUE or FALSE
          )
```

Where:

- The transportDomain is that of the destination of the message.

- The transportAddress is that of the destination of the message.

- The messageProcessingModel indicates which Message Processing Model the application wishes to use.

- The securityModel is the security model that the application wishes to use.

- The securityName is the security model independent name for the principal on whose behalf the application wishes the message to be generated.

- The securityLevel is the security level that the application wishes to use.

- The contextEngineID specifies the location of the management information it is requesting. Note that unless the request is being sent to a proxy, this value will usually be equal to the snmpEngineID value of the engine to which the request is being sent.

- The contextName specifies the local context name for the management information it is requesting.

- The pduVersion indicates the version of the PDU to be sent.

- The PDU is a value constructed by the command generator containing the management operation that the command generator wishes to perform.

- The expectResponse argument indicates that a response is expected.

The result of the sendPdu interface indicates whether the PDU was successfully sent. If it was successfully sent, the returned value will be a sendPduHandle. The command generator should store the sendPduHandle so that it can correlate a response to the original request.

The Dispatcher is responsible for delivering the response to a particular request to the correct command generator application. The abstract service interface used is:

```
   processResponsePdu(              -- process Response PDU
     IN   messageProcessingModel    -- typically, SNMP version
     IN   securityModel             -- Security Model in use
     IN   securityName              -- on behalf of this principal
     IN   securityLevel             -- Level of Security
     IN   contextEngineID           -- data from/at this SNMP entity
     IN   contextName               -- data from/in this context
     IN   pduVersion                -- the version of the PDU
     IN   PDU                       -- SNMP Protocol Data Unit
     IN   statusInformation         -- success or errorIndication
     IN   sendPduHandle             -- handle from sendPdu
          )
```

Where:

- The messageProcessingModel is the value from the received response.

- The securityModel is the value from the received response.

- The securityName is the value from the received response.

- The securityLevel is the value from the received response.

- The contextEngineID is the value from the received response.

- The contextName is the value from the received response.

- The pduVersion indicates the version of the PDU in the received response.

- The PDU is the value from the received response.

- The statusInformation indicates success or failure in receiving the response.

- The sendPduHandle is the value returned by the sendPdu call which generated the original request to which this is a response.

The procedure when a command generator receives a message is as follows:

(1) If the received values of messageProcessingModel, securityModel, securityName, contextEngineID, contextName, and pduVersion are not all equal to the values used in the original request, the response is discarded.

(2) The operation type, request-id, error-status, error-index, and variable-bindings are extracted from the PDU and saved. If the request-id is not equal to the value used in the original request, the response is discarded.

(3) At this point, it is up to the application to take an appropriate action. The specific action is implementation dependent. If the statusInformation indicates that the request failed, an appropriate action might be to attempt to transmit the request again, or to notify the person operating the application that a failure occurred.

3.2. Command Responder Applications

Before a command responder application can process messages, it must first associate itself with an SNMP engine. The abstract service interface used for this purpose is:

```
statusInformation =       -- success or errorIndication
 registerContextEngineID(
 IN   contextEngineID     -- take responsibility for this one
 IN   pduType             -- the pduType(s) to be registered
      )
```

Where:

- The statusInformation indicates success or failure of the registration attempt.

- The contextEngineID is equal to the snmpEngineID of the SNMP engine with which the command responder is registering.

- The pduType indicates a Read-Class and/or Write-Class PDU.

Note that if another command responder application is already registered with an SNMP engine, any further attempts to register with the same contextEngineID and pduType will be denied. This implies that separate command responder applications could register separately for the various pdu types. However, in practice this is undesirable, and only a single command responder application should be registered with an SNMP engine at any given time.

A command responder application can disassociate with an SNMP engine using the following abstract service interface:

```
unregisterContextEngineID(
  IN   contextEngineID     -- give up responsibility for this one
  IN   pduType             -- the pduType(s) to be unregistered
       )
```

Where:

- The contextEngineID is equal to the snmpEngineID of the SNMP engine with which the command responder is cancelling the registration.

- The pduType indicates a Read-Class and/or Write-Class PDU.

Once the command responder has registered with the SNMP engine, it waits to receive SNMP messages. The abstract service interface used for receiving messages is:

```
processPdu(                       -- process Request/Notification PDU
  IN   messageProcessingModel     -- typically, SNMP version
  IN   securityModel              -- Security Model in use
  IN   securityName               -- on behalf of this principal
  IN   securityLevel              -- Level of Security
  IN   contextEngineID            -- data from/at this SNMP entity
  IN   contextName                -- data from/in this context
  IN   pduVersion                 -- the version of the PDU
  IN   PDU                        -- SNMP Protocol Data Unit
  IN   maxSizeResponseScopedPDU -- maximum size of the Response PDU
  IN   stateReference             -- reference to state information
       )                          -- needed when sending a response
```

Where:

- The messageProcessingModel indicates which Message Processing Model received and processed the message.

- The securityModel is the value from the received message.

- The securityName is the value from the received message.

- The securityLevel is the value from the received message.

- The contextEngineID is the value from the received message.

- The contextName is the value from the received message.

- The pduVersion indicates the version of the PDU in the received message.

- The PDU is the value from the received message.

- The maxSizeResponseScopedPDU is the maximum allowable size of a ScopedPDU containing a Response PDU (based on the maximum message size that the originator of the message can accept).

- The stateReference is a value which references cached information about each received request message. This value must be returned to the Dispatcher in order to generate a response.

The procedure when a message is received is as follows:

(1) The operation type is determined from the ASN.1 tag value associated with the PDU parameter. The operation type should always be one of the types previously registered by the application.

(2) The request-id is extracted from the PDU and saved.

(3) Any PDU type specific parameters are extracted from the PDU and saved (for example, if the PDU type is an SNMPv2 GetBulk PDU, the non-repeaters and max-repetitions values are extracted).

(4) The variable-bindings are extracted from the PDU and saved.

(5) The management operation represented by the PDU type is performed with respect to the relevant MIB view within the context named by the contextName (for an SNMPv2 PDU type, the operation is performed according to the procedures set forth in [RFC1905]). The relevant MIB view is determined by the securityLevel, securityModel, contextName, securityName, and the class of the PDU type. To determine whether a particular object instance is within the relevant MIB view, the following abstract service interface is called:

```
   statusInformation =      -- success or errorIndication
     isAccessAllowed(
     IN   securityModel     -- Security Model in use
     IN   securityName      -- principal who wants to access
     IN   securityLevel     -- Level of Security
     IN   viewType          -- read, write, or notify view
     IN   contextName       -- context containing variableName
     IN   variableName      -- OID for the managed object
          )
```

Where:

- The securityModel is the value from the received message.

- The securityName is the value from the received message.

- The securityLevel is the value from the received message.

- The viewType indicates whether the PDU type is a Read-Class or Write-Class operation.

- The contextName is the value from the received message.

- The variableName is the object instance of the variable for which access rights are to be checked.

Normally, the result of the management operation will be a new PDU value, and processing will continue in step (6) below. However, at any time during the processing of the management operation:

- If the isAccessAllowed ASI returns a noSuchView, noAccessEntry, or noGroupName error, processing of the management operation is halted, a PDU value is constructed using the values from the originally received PDU, but replacing the error-status with an authorizationError code, and error-index value of 0, and control is passed to step (6) below.

- If the isAccessAllowed ASI returns an otherError, processing of the management operation is halted, a different PDU value is constructed using the values from the originally received PDU, but replacing the error-status with a genError code and the error-index with the index of the failed variable binding, and control is passed to step (6) below.

- If the isAccessAllowed ASI returns a noSuchContext error, processing of the management operation is halted, no result PDU is generated, the snmpUnknownContexts counter is incremented, and control is passed to step (6) below for generation of a report message.

- If the context named by the contextName parameter is unavailable, processing of the management operation is halted, no result PDU is generated, the snmpUnavailableContexts counter is incremented, and control is passed to step (6) below for generation of a report message.

(6) The Dispatcher is called to generate a response or report message. The abstract service interface is:

```
returnResponsePdu(
  IN   messageProcessingModel   -- typically, SNMP version
  IN   securityModel            -- Security Model in use
  IN   securityName             -- on behalf of this principal
  IN   securityLevel            -- same as on incoming request
  IN   contextEngineID          -- data from/at this SNMP entity
  IN   contextName              -- data from/in this context
  IN   pduVersion               -- the version of the PDU
  IN   PDU                      -- SNMP Protocol Data Unit
  IN   maxSizeResponseScopedPDU -- maximum size of the Response PDU
  IN   stateReference           -- reference to state information
                                -- as presented with the request
  IN   statusInformation        -- success or errorIndication
       )                        -- error counter OID/value if error
```

Where:

- The messageProcessingModel is the value from the processPdu call.

- The securityModel is the value from the processPdu call.

- The securityName is the value from the processPdu call.

- The securityLevel is the value from the processPdu call.

- The contextEngineID is the value from the processPdu call.

- The contextName is the value from the processPdu call.

- The pduVersion indicates the version of the PDU to be returned. If no result PDU was generated, the pduVersion is an undefined value.

- The PDU is the result generated in step (5) above. If no result PDU was generated, the PDU is an undefined value.

- The maxSizeResponseScopedPDU is a local value indicating the maximum size of a ScopedPDU that the application can accept.

- The stateReference is the value from the processPdu call.

- The statusInformation either contains an indication that no error occurred and that a response should be generated, or contains an indication that an error occurred along with the OID and counter value of the appropriate error counter object.

Note that a command responder application should always call the returnResponsePdu abstract service interface, even in the event of an error such as a resource allocation error. In the event of such an error, the PDU value passed to returnResponsePdu should contain appropriate values for errorStatus and errorIndex.

Note that the text above describes situations where the snmpUnknownContexts counter is incremented, and where the snmpUnavailableContexts counter is incremented. The difference between these is that the snmpUnknownContexts counter is incremented when a request is received for a context which is unknown to the SNMP entity. The snmpUnavailableContexts counter is incremented when a request is received for a context which is known to the SNMP entity, but is currently unavailable. Determining when a context is unavailable is implementation specific, and some implementations may never encounter this situation, and so may never increment the snmpUnavailableContexts counter.

3.3. Notification Originator Applications

A notification originator application generates SNMP messages containing Notification-Class PDUs (for example, SNMPv2-Trap PDUs or Inform PDUs). There is no requirement as to what specific types of Notification-Class PDUs a particular implementation must be capable of generating.

Notification originator applications require a mechanism for identifying the management targets to which notifications should be sent. The particular mechanism used is implementation dependent. However, if an implementation makes the configuration of management targets SNMP manageable, it MUST use the SNMP-TARGET-MIB module described in this document.

When a notification originator wishes to generate a notification, it must first determine in which context the information to be conveyed in the notification exists, i.e., it must determine the contextEngineID and contextName. It must then determine the set of management targets to which the notification should be sent. The application must also determine, for each management target, what specific PDU type the notification message should contain, and if it is to contain a Confirmed-Class PDU, the number of retries and retransmission algorithm.

The mechanism by which a notification originator determines this information is implementation dependent. Once the application has determined this information, the following procedure is performed for each management target:

(1) Any appropriate filtering mechanisms are applied to determine whether the notification should be sent to the management target. If such filtering mechanisms determine that the notification should not be sent, processing continues with the next management target. Otherwise,

(2) The appropriate set of variable-bindings is retrieved from local MIB instrumentation within the relevant MIB view. The relevant MIB view is determined by the securityLevel, securityModel, contextName, and securityName of the management target. To determine whether a particular object instance is within the relevant MIB view, the isAccessAllowed abstract service interface is used, in the same manner as described in the preceding section, except that the viewType indicates a Notification-Class operation. If the statusInformation returned by isAccessAllowed does not indicate accessAllowed, the notification is not sent to the management target.

(3) The NOTIFICATION-TYPE OBJECT IDENTIFIER of the notification (this is the value of the element of the variable bindings whose name is snmpTrapOID.0, i.e., the second variable binding) is checked using the isAccessAllowed abstract service interface, using the same parameters used in the preceding step. If the statusInformation returned by isAccessAllowed does not indicate accessAllowed, the notification is not sent to the management target.

(4) A PDU is constructed using a locally unique request-id value, a PDU type as determined by the implementation, an error-status and error-index value of 0, and the variable-bindings supplied previously in step (2).

(5) If the notification contains an Unconfirmed-Class PDU, the Dispatcher is called using the following abstract service interface:

```
statusInformation =             -- sendPduHandle if success
                                -- errorIndication if failure
  sendPdu(
  IN   transportDomain          -- transport domain to be used
  IN   transportAddress         -- destination network address
  IN   messageProcessingModel   -- typically, SNMP version
  IN   securityModel            -- Security Model to use
  IN   securityName             -- on behalf of this principal
  IN   securityLevel            -- Level of Security requested
  IN   contextEngineID          -- data from/at this entity
  IN   contextName              -- data from/in this context
  IN   pduVersion               -- the version of the PDU
  IN   PDU                      -- SNMP Protocol Data Unit
  IN   expectResponse           -- TRUE or FALSE
       )
```

Where:

- The transportDomain is that of the management target.

- The transportAddress is that of the management target.

- The messageProcessingModel is that of the management target.

- The securityModel is that of the management target.

- The securityName is that of the management target.

- The securityLevel is that of the management target.

- The contextEngineID is the value originally determined for the notification.

- The contextName is the value originally determined for the notification.

- The pduVersion is the version of the PDU to be sent.

- The PDU is the value constructed in step (4) above.

- The expectResponse argument indicates that no response is expected.

Otherwise,

(6) If the notification contains a Confirmed-Class PDU, then:

a) The Dispatcher is called using the sendPdu abstract service interface as described in step (5) above, except that the expectResponse argument indicates that a response is expected.

b) The application caches information about the management target.

c) If a response is received within an appropriate time interval from the transport endpoint of the management target, the notification is considered acknowledged and the cached information is deleted. Otherwise,

d) If a response is not received within an appropriate time period, or if a report indication is received, information about the management target is retrieved from the cache, and steps a) through d) are repeated. The number of times these steps are repeated is equal to the previously determined retry count. If this retry count is exceeded, the acknowledgement of the notification is considered to have failed, and processing of the notification for this management target is halted. Note that some report indications might be considered a failure. Such report indications should be interpreted to mean that the acknowledgement of the notification has failed, and that steps a) through d) need not be repeated.

Responses to Confirmed-Class PDU notifications will be received via the processResponsePdu abstract service interface.

To summarize, the steps that a notification originator follows when determining where to send a notification are:

- Determine the targets to which the notification should be sent.

- Apply any required filtering to the list of targets.

- Determine which targets are authorized to receive the notification.

3.4. Notification Receiver Applications

Notification receiver applications receive SNMP Notification messages from the Dispatcher. Before any messages can be received, the notification receiver must register with the Dispatcher using the registerContextEngineID abstract service interface. The parameters used are:

- The contextEngineID is an undefined 'wildcard' value. Notifications are delivered to a registered notification receiver regardless of the contextEngineID contained in the notification message.

- The pduType indicates the type of notifications that the application wishes to receive (for example, SNMPv2-Trap PDUs or Inform PDUs).

Once the notification receiver has registered with the Dispatcher, messages are received using the processPdu abstract service interface. Parameters are:

- The messageProcessingModel indicates which Message Processing Model received and processed the message.

- The securityModel is the value from the received message.

- The securityName is the value from the received message.

- The securityLevel is the value from the received message.

- The contextEngineID is the value from the received message.

- The contextName is the value from the received message.

- The pduVersion indicates the version of the PDU in the received message.

- The PDU is the value from the received message.

- The maxSizeResponseScopedPDU is the maximum allowable size of a ScopedPDU containing a Response PDU (based on the maximum message size that the originator of the message can accept).

- If the message contains an Unconfirmed-Class PDU, the stateReference is undefined and unused. Otherwise, the stateReference is a value which references cached information about the notification. This value must be returned to the Dispatcher in order to generate a response.

When an Unconfirmed-Class PDU is delivered to a notification receiver application, it first extracts the SNMP operation type, request-id, error-status, error-index, and variable-bindings from the PDU. After this, processing depends on the particular implementation.

When a Confirmed-Class PDU is received, the notification receiver application follows the following procedure:

(1) The PDU type, request-id, error-status, error-index, and variable-bindings are extracted from the PDU.

(2) A Response-Class PDU is constructed using the extracted request-id and variable-bindings, and with error-status and error-index both set to 0.

(3) The Dispatcher is called to generate a response message using the returnResponsePdu abstract service interface. Parameters are:

- The messageProcessingModel is the value from the processPdu call.

- The securityModel is the value from the processPdu call.

- The securityName is the value from the processPdu call.

- The securityLevel is the value from the processPdu call.

- The contextEngineID is the value from the processPdu call.

- The contextName is the value from the processPdu call.

- The pduVersion indicates the version of the PDU to be returned.

- The PDU is the result generated in step (2) above.

- The maxSizeResponseScopedPDU is a local value indicating the maximum size of a ScopedPDU that the application can accept.

- The stateReference is the value from the processPdu call.

- The statusInformation indicates that no error occurred and that a response should be generated.

(4) After this, processing depends on the particular implementation.

3.5. Proxy Forwarder Applications

A proxy forwarder application deals with forwarding SNMP messages. There are four basic types of messages which a proxy forwarder application may need to forward. These are grouped according to the class of PDU type contained in a message. The four basic types of messages are:

- Those containing Read-Class or Write-Class PDU types (for example, Get, GetNext, GetBulk, and Set PDU types). These deal with requesting or modifying information located within a particular context.

- Those containing Notification-Class PDU types (for example, SNMPv2-Trap and Inform PDU types). These deal with notifications concerning information located within a particular context.

- Those containing a Response-Class PDU type. Forwarding of Response-Class PDUs always occurs as a result of receiving a response to a previously forwarded message.

- Those containing Internal-Class PDU types (for example, a Report PDU). Forwarding of Internal-Class PDU types always occurs as a result of receiving an Internal-Class PDU in response to a previously forwarded message.

For the first type, the proxy forwarder's role is to deliver a request for management information to an SNMP engine which is "closer" or "downstream in the path" to the SNMP engine which has access to that information, and to deliver the response containing the information back to the SNMP engine from which the request was received. The context information in a request is used to determine which SNMP engine has access to the requested information, and this is used to determine where and how to forward the request.

For the second type, the proxy forwarder's role is to determine which SNMP engines should receive notifications about management information from a particular location. The context information in a notification message determines the location to which the information contained in the notification applies. This is used to determine which SNMP engines should receive notification about this information.

For the third type, the proxy forwarder's role is to determine which previously forwarded request or notification (if any) the response matches, and to forward the response back to the initiator of the request or notification.

For the fourth type, the proxy forwarder's role is to determine which previously forwarded request or notification (if any) the Internal-Class PDU matches, and to forward the Internal-Class PDU back to the initiator of the request or notification.

When forwarding messages, a proxy forwarder application must perform a translation of incoming management target information into outgoing management target information. How this translation is performed is implementation specific. In many cases, this will be driven by a preconfigured translation table. If a proxy forwarder application makes the contents of this table SNMP manageable, it MUST use the SNMP-PROXY-MIB module defined in this document.

3.5.1. Request Forwarding

There are two phases for request forwarding. First, the incoming request needs to be passed through the proxy application. Then, the resulting response needs to be passed back. These phases are described in the following two sections.

3.5.1.1. Processing an Incoming Request

A proxy forwarder application that wishes to forward request messages must first register with the Dispatcher using the registerContextEngineID abstract service interface. The proxy forwarder must register each contextEngineID for which it wishes to forward messages, as well as for each pduType. Note that as the configuration of a proxy forwarder is changed, the particular contextEngineID values for which it is forwarding may change. The proxy forwarder should call the registerContextEngineID and unregisterContextEngineID abstract service interfaces as needed to reflect its current configuration.

A proxy forwarder application should never attempt to register a value of contextEngineID which is equal to the snmpEngineID of the SNMP engine to which the proxy forwarder is associated.

Once the proxy forwarder has registered for the appropriate contextEngineID values, it can start processing messages. The following procedure is used:

(1) A message is received using the processPdu abstract service interface. The incoming management target information received from the processPdu interface is translated into outgoing management target information. Note that this translation may vary for different values of contextEngineID and/or contextName. The translation should result in a single management target.

(2) If appropriate outgoing management target information cannot be found, the proxy forwarder increments the snmpProxyDrops counter [RFC1907], and then calls the Dispatcher using the returnResponsePdu abstract service interface. Parameters are:

- The messageProcessingModel is the value from the processPdu call.

- The securityModel is the value from the processPdu call.

- The securityName is the value from the processPdu call.

- The securityLevel is the value from the processPdu call.

- The contextEngineID is the value from the processPdu call.

- The contextName is the value from the processPdu call.

- The pduVersion is the value from the processPdu call.

- The PDU is an undefined value.

- The maxSizeResponseScopedPDU is a local value indicating the maximum size of a ScopedPDU that the application can accept.

- The stateReference is the value from the processPdu call.

- The statusInformation indicates that an error occurred and includes the OID and value of the snmpProxyDrops object.

Processing of the message stops at this point. Otherwise,

(3) A new PDU is constructed. A unique value of request-id should be used in the new PDU (this value will enable a subsequent response message to be correlated with this request). The remainder of the new PDU is identical to the received PDU, unless the incoming SNMP version and the outgoing SNMP version support different PDU versions, in which case the proxy forwarder may need to perform a translation on the PDU. (A method for performing such a translation is described in [RFC2576].)

(4) The proxy forwarder calls the Dispatcher to generate the forwarded message, using the sendPdu abstract service interface. The parameters are:

- The transportDomain is that of the outgoing management target.

- The transportAddress is that of the outgoing management target.

- The messageProcessingModel is that of the outgoing management target.

- The securityModel is that of the outgoing management target.

- The securityName is that of the outgoing management target.

- The securityLevel is that of the outgoing management target.

- The contextEngineID is the value from the processPdu call.

- The contextName is the value from the processPdu call.

- The pduVersion is the version of the PDU to be sent.

- The PDU is the value constructed in step (3) above.

- The expectResponse argument indicates that a response is expected. If the sendPdu call is unsuccessful, the proxy forwarder performs the steps described in (2) above. Otherwise:

(5) The proxy forwarder caches the following information in order to match an incoming response to the forwarded request:

- The sendPduHandle returned from the call to sendPdu,

- The request-id from the received PDU.

- The contextEngineID,

- The contextName,

- The stateReference,

- The incoming management target information,

- The outgoing management information,

- Any other information needed to match an incoming response to the forwarded request.

If this information cannot be cached (possibly due to a lack of resources), the proxy forwarder performs the steps described in (2) above. Otherwise:

(6) Processing of the request stops until a response to the forwarded request is received, or until an appropriate time interval has expired. If this time interval expires before a response has been received, the cached information about this request is removed.

3.5.1.2. Processing an Incoming Response

A proxy forwarder follows the following procedure when an incoming response is received:

(1) The incoming response is received using the processResponsePdu interface. The proxy forwarder uses the received parameters to locate an entry in its cache of pending forwarded requests. This is done by matching the received parameters with the cached values of sendPduHandle, contextEngineID, contextName, outgoing management target information, and the request-id contained in the received PDU (the proxy forwarder must extract the request-id for this purpose). If an appropriate cache entry cannot be found, processing of the response is halted. Otherwise:

(2) The cache information is extracted, and removed from the cache.

(3) A new Response-Class PDU is constructed, using the request-id value from the original forwarded request (as extracted from the cache). All other values are identical to those in the received Response-Class PDU, unless the incoming SNMP version and the outgoing SNMP version support different PDU versions, in which case the proxy forwarder may need to perform a translation on the PDU. (A method for performing such a translation is described in [RFC2576].)

(4) The proxy forwarder calls the Dispatcher using the returnResponsePdu abstract service interface. Parameters are:

- The messageProcessingModel indicates the Message Processing Model by which the original incoming message was processed.

- The securityModel is that of the original incoming management target extracted from the cache.

- The securityName is that of the original incoming management target extracted from the cache.

- The securityLevel is that of the original incoming management target extracted from the cache.

- The contextEngineID is the value extracted from the cache.

- The contextName is the value extracted from the cache.

- The pduVersion indicates the version of the PDU to be returned.

- The PDU is the (possibly translated) Response PDU.

- The maxSizeResponseScopedPDU is a local value indicating the maximum size of a ScopedPDU that the application can accept.

- The stateReference is the value extracted from the cache.

- The statusInformation indicates that no error occurred and that a Response PDU message should be generated.

3.5.1.3. Processing an Incoming Internal-Class PDU

A proxy forwarder follows the following procedure when an incoming Internal-Class PDU is received:

(1) The incoming Internal-Class PDU is received using the processResponsePdu interface. The proxy forwarder uses the received parameters to locate an entry in its cache of pending forwarded requests. This is done by matching the received parameters with the cached values of sendPduHandle. If an appropriate cache entry cannot be found, processing of the Internal-Class PDU is halted. Otherwise:

(2) The cache information is extracted, and removed from the cache.

(3) If the original incoming management target information indicates an SNMP version which does not support Report PDUs, processing of the Internal-Class PDU is halted.

(4) The proxy forwarder calls the Dispatcher using the returnResponsePdu abstract service interface. Parameters are:

- The messageProcessingModel indicates the Message Processing Model by which the original incoming message was processed.

- The securityModel is that of the original incoming management target extracted from the cache.

- The securityName is that of the original incoming management target extracted from the cache.

- The securityLevel is that of the original incoming management target extracted from the cache.

- The contextEngineID is the value extracted from the cache.

- The contextName is the value extracted from the cache.

- The pduVersion indicates the version of the PDU to be returned.

- The PDU is unused.

- The maxSizeResponseScopedPDU is a local value indicating the maximum size of a ScopedPDU that the application can accept.

- The stateReference is the value extracted from the cache.

- The statusInformation contains values specific to the Internal-Class PDU type (for example, for a Report PDU, the statusInformation contains the contextEngineID, contextName, counter OID, and counter value received in the incoming Report PDU).

3.5.2. Notification Forwarding

A proxy forwarder receives notifications in the same manner as a notification receiver application, using the processPdu abstract service interface. The following procedure is used when a notification is received:

(1) The incoming management target information received from the processPdu interface is translated into outgoing management target information. Note that this translation may vary for different values of contextEngineID and/or contextName. The translation may result in multiple management targets.

(2) If appropriate outgoing management target information cannot be found and the notification was an Unconfirmed-Class PDU, processing of the notification is halted. If appropriate outgoing management target information cannot be found and the notification was a Confirmed-Class PDU, the proxy forwarder increments the snmpProxyDrops object, and calls the Dispatcher using the returnResponsePdu abstract service interface. The parameters are:

- The messageProcessingModel is the value from the processPdu call.

- The securityModel is the value from the processPdu call.

- The securityName is the value from the processPdu call.

- The securityLevel is the value from the processPdu call.

- The contextEngineID is the value from the processPdu call.

- The contextName is the value from the processPdu call.

- The pduVersion is the value from the processPdu call.

- The PDU is an undefined and unused value.

- The maxSizeResponseScopedPDU is a local value indicating the maximum size of a ScopedPDU that the application can accept.

- The stateReference is the value from the processPdu call.

- The statusInformation indicates that an error occurred and that a Report message should be generated.

  Processing of the message stops at this point. Otherwise,

(3) The proxy forwarder generates a notification using the procedures described in the preceding section on Notification Originators, with the following exceptions:

- The contextEngineID and contextName values from the original received notification are used.

- The outgoing management targets previously determined are used.

- No filtering mechanisms are applied.

- The variable-bindings from the original received notification are used, rather than retrieving variable-bindings from local MIB instrumentation. In particular, no access-control is applied to these variable-bindings, nor to the value of the variable-binding containing snmpTrapOID.0.

- If the original notification contains a Confirmed-Class PDU, then any outgoing management targets for which the outgoing SNMP version does not support any PDU types that are both Notification-Class and Confirmed-Class PDUs will not be used when generating the forwarded notifications.

- If, for any of the outgoing management targets, the incoming SNMP version and the outgoing SNMP version support different PDU versions, the proxy forwarder may need to perform a translation on the PDU. (A method for performing such a translation is described in [RFC2576].)

(4) If the original received notification contains an Unconfirmed-Class PDU, processing of the notification is now completed. Otherwise, the original received notification must contain Confirmed-Class PDU, and processing continues.

(5) If the forwarded notifications included any Confirmed-Class PDUs, processing continues when the procedures described in the section for Notification Originators determine that either:

- None of the generated notifications containing Confirmed-Class PDUs have been successfully acknowledged within the longest of the time intervals, in which case processing of the original notification is halted, or,

- At least one of the generated notifications containing Confirmed-Class PDUs is successfully acknowledged, in which case a response to the original received notification containing an Confirmed-Class PDU is generated as described in the following steps.

(6) A Response-Class PDU is constructed, using the values of request-id and variable-bindings from the original received Notification-Class PDU, and error-status and error-index values of 0.

(7) The Dispatcher is called using the returnResponsePdu abstract service interface. Parameters are:

- The messageProcessingModel is the value from the processPdu call.

- The securityModel is the value from the processPdu call.

- The securityName is the value from the processPdu call.

- The securityLevel is the value from the processPdu call.

- The contextEngineID is the value from the processPdu call.

- The contextName is the value from the processPdu call.

- The pduVersion indicates the version of the PDU constructed in step (6) above.

- The PDU is the value constructed in step (6) above.

- The maxSizeResponseScopedPDU is a local value indicating the maximum size of a ScopedPDU that the application can accept.

- The stateReference is the value from the processPdu call.

- The statusInformation indicates that no error occurred and that a Response-Class PDU message should be generated.

4. The Structure of the MIB Modules

There are three separate MIB modules described in this document, the management target MIB, the notification MIB, and the proxy MIB. The following sections describe the structure of these three MIB modules.

The use of these MIBs by particular types of applications is described later in this document:

- The use of the management target MIB and the notification MIB in notification originator applications is described in section 5.

- The use of the notification MIB for filtering notifications in notification originator applications is described in section 6.

- The use of the management target MIB and the proxy MIB in proxy forwarding applications is described in section 7.

4.1. The Management Target MIB Module

The SNMP-TARGET-MIB module contains objects for defining management targets. It consists of two tables and conformance/compliance statements.

The first table, the snmpTargetAddrTable, contains information about transport domains and addresses. It also contains an object, snmpTargetAddrTagList, which provides a mechanism for grouping entries.

The second table, the snmpTargetParamsTable, contains information about SNMP version and security information to be used when sending messages to particular transport domains and addresses.

The Management Target MIB is intended to provide a general-purpose mechanism for specifying transport address, and for specifying parameters of SNMP messages generated by an SNMP entity. It is used within this document for generation of notifications and for proxy forwarding. However, it may be used for other purposes. If another document makes use of this MIB, that document is responsible for specifying how it is used. For example, [RFC2576] uses this MIB for source address validation of SNMPv1 messages.

4.1.1. Tag Lists

The snmpTargetAddrTagList object is used for grouping entries in the snmpTargetAddrTable. The value of this object contains a list of tag values which are used to select target addresses to be used for a particular operation.

A tag value, which may also be used in MIB objects other than snmpTargetAddrTagList, is an arbitrary string of octets, but may not contain a delimiter character. Delimiter characters are defined to be one of the following characters:

- An ASCII space character (0x20).

- An ASCII TAB character (0x09).

- An ASCII carriage return (CR) character (0x0D).

- An ASCII line feed (LF) character (0x0A).

In addition, a tag value within a tag list may not have a zero length. Generally, a particular MIB object may contain either

- a zero-length octet string representing an empty list, or

- a single tag value, in which case the value of the MIB object may not contain a delimiter character, or

- a list of tag values, separated by single delimiter characters.

  For a list of tag values, these constraints imply certain restrictions on the value of a MIB object:

- There cannot be a leading or trailing delimiter character.

- There cannot be multiple adjacent delimiter characters.

4.1.2. Definitions

```
SNMP-TARGET-MIB DEFINITIONS ::= BEGIN

IMPORTS
    MODULE-IDENTITY,
    OBJECT-TYPE,
    snmpModules,
    Counter32,
    Integer32
        FROM SNMPv2-SMI

    TEXTUAL-CONVENTION,
    TDomain,
    TAddress,
    TimeInterval,
    RowStatus,
    StorageType,
```

```
       TestAndIncr
           FROM SNMPv2-TC

       SnmpSecurityModel,
       SnmpMessageProcessingModel,
       SnmpSecurityLevel,
       SnmpAdminString
           FROM SNMP-FRAMEWORK-MIB

       MODULE-COMPLIANCE,
       OBJECT-GROUP
           FROM SNMPv2-CONF;

   snmpTargetMIB MODULE-IDENTITY
       LAST-UPDATED "200210140000Z"
       ORGANIZATION "IETF SNMPv3 Working Group"
       CONTACT-INFO
           "WG-email:   snmpv3@lists.tislabs.com
            Subscribe:  majordomo@lists.tislabs.com
                        In message body:  subscribe snmpv3

            Co-Chair:   Russ Mundy
                        Network Associates Laboratories
            Postal:     15204 Omega Drive, Suite 300
                        Rockville, MD 20850-4601
                        USA
            EMail:      mundy@tislabs.com
            Phone:      +1 301-947-7107

            Co-Chair:   David Harrington
                        Enterasys Networks
            Postal:     35 Industrial Way
                        P. O. Box 5004
                        Rochester, New Hampshire 03866-5005
                        USA
            EMail:      dbh@enterasys.com
            Phone:      +1 603-337-2614

            Co-editor:  David B. Levi
                        Nortel Networks
            Postal:     3505 Kesterwood Drive
                        Knoxville, Tennessee 37918
            EMail:      dlevi@nortelnetworks.com
            Phone:      +1 865 686 0432

            Co-editor:  Paul Meyer
                        Secure Computing Corporation
            Postal:     2675 Long Lake Road
```

```
                    Roseville, Minnesota 55113
            EMail:  paul_meyer@securecomputing.com
            Phone:  +1 651 628 1592

            Co-editor:  Bob Stewart
                        Retired"
        DESCRIPTION
            "This MIB module defines MIB objects which provide
             mechanisms to remotely configure the parameters used
             by an SNMP entity for the generation of SNMP messages.

             Copyright (C) The Internet Society (2002). This
             version of this MIB module is part of RFC 3413;
             see the RFC itself for full legal notices.
            "
        REVISION    "200210140000Z"             -- 14 October 2002
        DESCRIPTION "Fixed DISPLAY-HINTS for UTF-8 strings, fixed hex
                     value of LF characters, clarified meaning of zero
                     length tag values, improved tag list examples.
                     Published as RFC 3413."
        REVISION    "199808040000Z"             -- 4 August 1998
        DESCRIPTION "Clarifications, published as
                     RFC 2573."
        REVISION    "199707140000Z"             -- 14 July 1997
        DESCRIPTION "The initial revision, published as RFC2273."
        ::= { snmpModules 12 }

    snmpTargetObjects       OBJECT IDENTIFIER ::= { snmpTargetMIB 1 }
    snmpTargetConformance   OBJECT IDENTIFIER ::= { snmpTargetMIB 3 }

    SnmpTagValue ::= TEXTUAL-CONVENTION
        DISPLAY-HINT "255t"
        STATUS       current
        DESCRIPTION
            "An octet string containing a tag value.
             Tag values are preferably in human-readable form.

             To facilitate internationalization, this information
             is represented using the ISO/IEC IS 10646-1 character
             set, encoded as an octet string using the UTF-8
             character encoding scheme described in RFC 2279.

             Since additional code points are added by amendments
             to the 10646 standard from time to time,
             implementations must be prepared to encounter any code
             point from 0x00000000 to 0x7fffffff.

             The use of control codes should be avoided, and certain
```

control codes are not allowed as described below.

For code points not directly supported by user interface hardware or software, an alternative means of entry and display, such as hexadecimal, may be provided.

For information encoded in 7-bit US-ASCII, the UTF-8 representation is identical to the US-ASCII encoding.

Note that when this TC is used for an object that is used or envisioned to be used as an index, then a SIZE restriction must be specified so that the number of sub-identifiers for any object instance does not exceed the limit of 128, as defined by [RFC1905].

An object of this type contains a single tag value which is used to select a set of entries in a table.

A tag value is an arbitrary string of octets, but may not contain a delimiter character. Delimiter characters are defined to be one of the following:

- An ASCII space character (0x20).
- An ASCII TAB character (0x09).
- An ASCII carriage return (CR) character (0x0D).
- An ASCII line feed (LF) character (0x0A).

Delimiter characters are used to separate tag values in a tag list. An object of this type may only contain a single tag value, and so delimiter characters are not allowed in a value of this type.

Note that a tag value of 0 length means that no tag is defined. In other words, a tag value of 0 length would never match anything in a tag list, and would never select any table entries.

Some examples of valid tag values are:

- 'acme'
- 'router'
- 'host'

```
            The use of a tag value to select table entries is
            application and MIB specific."
        SYNTAX       OCTET STRING (SIZE (0..255))

    SnmpTagList ::= TEXTUAL-CONVENTION
        DISPLAY-HINT "255t"
        STATUS       current
        DESCRIPTION
            "An octet string containing a list of tag values.
             Tag values are preferably in human-readable form.

             To facilitate internationalization, this information
             is represented using the ISO/IEC IS 10646-1 character
             set, encoded as an octet string using the UTF-8
             character encoding scheme described in RFC 2279.

             Since additional code points are added by amendments
             to the 10646 standard from time to time,
             implementations must be prepared to encounter any code
             point from 0x00000000 to 0x7fffffff.

             The use of control codes should be avoided, except as
             described below.

             For code points not directly supported by user
             interface hardware or software, an alternative means
             of entry and display, such as hexadecimal, may be
             provided.

             For information encoded in 7-bit US-ASCII, the UTF-8
             representation is identical to the US-ASCII encoding.

             An object of this type contains a list of tag values
             which are used to select a set of entries in a table.

             A tag value is an arbitrary string of octets, but
             may not contain a delimiter character.  Delimiter
             characters are defined to be one of the following:

                 -  An ASCII space character (0x20).

                 -  An ASCII TAB character (0x09).

                 -  An ASCII carriage return (CR) character (0x0D).

                 -  An ASCII line feed (LF) character (0x0A).

             Delimiter characters are used to separate tag values
```

```
            in a tag list.  Only a single delimiter character may
            occur between two tag values.  A tag value may not
            have a zero length.  These constraints imply certain
            restrictions on the contents of this object:

                - There cannot be a leading or trailing delimiter
                  character.

                - There cannot be multiple adjacent delimiter
                  characters.

            Some examples of valid tag lists are:

                - ''                        -- an empty list

                - 'acme'                    -- list of one tag

                - 'host router bridge'      -- list of several tags

            Note that although a tag value may not have a length of
            zero, an empty string is still valid.  This indicates
            an empty list (i.e. there are no tag values in the list).

            The use of the tag list to select table entries is
            application and MIB specific.  Typically, an application
            will provide one or more tag values, and any entry
            which contains some combination of these tag values
            will be selected."
        SYNTAX       OCTET STRING (SIZE (0..255))

    --
    --
    -- The snmpTargetObjects group
    --
    --

    snmpTargetSpinLock OBJECT-TYPE
        SYNTAX      TestAndIncr
        MAX-ACCESS  read-write
        STATUS      current
        DESCRIPTION
            "This object is used to facilitate modification of table
            entries in the SNMP-TARGET-MIB module by multiple
            managers.  In particular, it is useful when modifying
            the value of the snmpTargetAddrTagList object.

            The procedure for modifying the snmpTargetAddrTagList
            object is as follows:
```

```
                1.  Retrieve the value of snmpTargetSpinLock and
                    of snmpTargetAddrTagList.

                2.  Generate a new value for snmpTargetAddrTagList.

                3.  Set the value of snmpTargetSpinLock to the
                    retrieved value, and the value of
                    snmpTargetAddrTagList to the new value.  If
                    the set fails for the snmpTargetSpinLock
                    object, go back to step 1."
        ::= { snmpTargetObjects 1 }

    snmpTargetAddrTable OBJECT-TYPE
        SYNTAX      SEQUENCE OF SnmpTargetAddrEntry
        MAX-ACCESS  not-accessible
        STATUS      current
        DESCRIPTION
            "A table of transport addresses to be used in the generation
             of SNMP messages."
        ::= { snmpTargetObjects 2 }

    snmpTargetAddrEntry OBJECT-TYPE
        SYNTAX      SnmpTargetAddrEntry
        MAX-ACCESS  not-accessible
        STATUS      current
        DESCRIPTION
            "A transport address to be used in the generation
             of SNMP operations.

             Entries in the snmpTargetAddrTable are created and
             deleted using the snmpTargetAddrRowStatus object."
        INDEX { IMPLIED snmpTargetAddrName }
        ::= { snmpTargetAddrTable 1 }

    SnmpTargetAddrEntry ::= SEQUENCE {
        snmpTargetAddrName          SnmpAdminString,
        snmpTargetAddrTDomain       TDomain,
        snmpTargetAddrTAddress      TAddress,
        snmpTargetAddrTimeout       TimeInterval,
        snmpTargetAddrRetryCount    Integer32,
        snmpTargetAddrTagList       SnmpTagList,
        snmpTargetAddrParams        SnmpAdminString,
        snmpTargetAddrStorageType   StorageType,
        snmpTargetAddrRowStatus     RowStatus
    }

    snmpTargetAddrName OBJECT-TYPE
        SYNTAX      SnmpAdminString (SIZE(1..32))
```

```
        MAX-ACCESS  not-accessible
        STATUS      current
        DESCRIPTION
            "The locally arbitrary, but unique identifier associated
             with this snmpTargetAddrEntry."
        ::= { snmpTargetAddrEntry 1 }

    snmpTargetAddrTDomain OBJECT-TYPE
        SYNTAX      TDomain
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "This object indicates the transport type of the address
             contained in the snmpTargetAddrTAddress object."
        ::= { snmpTargetAddrEntry 2 }

    snmpTargetAddrTAddress OBJECT-TYPE
        SYNTAX      TAddress
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "This object contains a transport address.  The format of
             this address depends on the value of the
             snmpTargetAddrTDomain object."
        ::= { snmpTargetAddrEntry 3 }

    snmpTargetAddrTimeout OBJECT-TYPE
        SYNTAX      TimeInterval
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "This object should reflect the expected maximum round
             trip time for communicating with the transport address
             defined by this row.  When a message is sent to this
             address, and a response (if one is expected) is not
             received within this time period, an implementation
             may assume that the response will not be delivered.

             Note that the time interval that an application waits
             for a response may actually be derived from the value
             of this object.  The method for deriving the actual time
             interval is implementation dependent.  One such method
             is to derive the expected round trip time based on a
             particular retransmission algorithm and on the number
             of timeouts which have occurred.  The type of message may
             also be considered when deriving expected round trip
             times for retransmissions.  For example, if a message is
             being sent with a securityLevel that indicates both
```

```
            authentication and privacy, the derived value may be
            increased to compensate for extra processing time spent
            during authentication and encryption processing."
        DEFVAL { 1500 }
        ::= { snmpTargetAddrEntry 4 }

    snmpTargetAddrRetryCount OBJECT-TYPE
        SYNTAX      Integer32 (0..255)
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "This object specifies a default number of retries to be
            attempted when a response is not received for a generated
            message.  An application may provide its own retry count,
            in which case the value of this object is ignored."
        DEFVAL { 3 }
        ::= { snmpTargetAddrEntry 5 }

    snmpTargetAddrTagList OBJECT-TYPE
        SYNTAX      SnmpTagList
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "This object contains a list of tag values which are
            used to select target addresses for a particular
            operation."
        DEFVAL { "" }
        ::= { snmpTargetAddrEntry 6 }

    snmpTargetAddrParams OBJECT-TYPE
        SYNTAX      SnmpAdminString (SIZE(1..32))
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "The value of this object identifies an entry in the
            snmpTargetParamsTable.  The identified entry
            contains SNMP parameters to be used when generating
            messages to be sent to this transport address."
        ::= { snmpTargetAddrEntry 7 }

    snmpTargetAddrStorageType OBJECT-TYPE
        SYNTAX      StorageType
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "The storage type for this conceptual row.
            Conceptual rows having the value 'permanent' need not
            allow write-access to any columnar objects in the row."
```

```
        DEFVAL { nonVolatile }
        ::= { snmpTargetAddrEntry 8 }

    snmpTargetAddrRowStatus OBJECT-TYPE
        SYNTAX      RowStatus
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "The status of this conceptual row.

             To create a row in this table, a manager must
             set this object to either createAndGo(4) or
             createAndWait(5).

             Until instances of all corresponding columns are
             appropriately configured, the value of the
             corresponding instance of the snmpTargetAddrRowStatus
             column is 'notReady'.

             In particular, a newly created row cannot be made
             active until the corresponding instances of
             snmpTargetAddrTDomain, snmpTargetAddrTAddress, and
             snmpTargetAddrParams have all been set.

             The following objects may not be modified while the
             value of this object is active(1):
                 - snmpTargetAddrTDomain
                 - snmpTargetAddrTAddress
             An attempt to set these objects while the value of
             snmpTargetAddrRowStatus is active(1) will result in
             an inconsistentValue error."
        ::= { snmpTargetAddrEntry 9 }

    snmpTargetParamsTable OBJECT-TYPE
        SYNTAX      SEQUENCE OF SnmpTargetParamsEntry
        MAX-ACCESS  not-accessible
        STATUS      current
        DESCRIPTION
            "A table of SNMP target information to be used
             in the generation of SNMP messages."
        ::= { snmpTargetObjects 3 }

    snmpTargetParamsEntry OBJECT-TYPE
        SYNTAX      SnmpTargetParamsEntry
        MAX-ACCESS  not-accessible
        STATUS      current
        DESCRIPTION
            "A set of SNMP target information.
```

```
            Entries in the snmpTargetParamsTable are created and
            deleted using the snmpTargetParamsRowStatus object."
        INDEX { IMPLIED snmpTargetParamsName }
        ::= { snmpTargetParamsTable 1 }

    SnmpTargetParamsEntry ::= SEQUENCE {
        snmpTargetParamsName           SnmpAdminString,
        snmpTargetParamsMPModel        SnmpMessageProcessingModel,
        snmpTargetParamsSecurityModel  SnmpSecurityModel,
        snmpTargetParamsSecurityName   SnmpAdminString,
        snmpTargetParamsSecurityLevel  SnmpSecurityLevel,
        snmpTargetParamsStorageType    StorageType,
        snmpTargetParamsRowStatus      RowStatus
    }

    snmpTargetParamsName OBJECT-TYPE
        SYNTAX      SnmpAdminString (SIZE(1..32))
        MAX-ACCESS  not-accessible
        STATUS      current
        DESCRIPTION
            "The locally arbitrary, but unique identifier associated
             with this snmpTargetParamsEntry."
        ::= { snmpTargetParamsEntry 1 }

    snmpTargetParamsMPModel OBJECT-TYPE
        SYNTAX      SnmpMessageProcessingModel
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "The Message Processing Model to be used when generating
             SNMP messages using this entry."
        ::= { snmpTargetParamsEntry 2 }

    snmpTargetParamsSecurityModel OBJECT-TYPE
        SYNTAX      SnmpSecurityModel (1..2147483647)
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "The Security Model to be used when generating SNMP
              messages using this entry.  An implementation may
              choose to return an inconsistentValue error if an
              attempt is made to set this variable to a value
              for a security model which the implementation does
              not support."
        ::= { snmpTargetParamsEntry 3 }

    snmpTargetParamsSecurityName OBJECT-TYPE
        SYNTAX      SnmpAdminString
```

```
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "The securityName which identifies the Principal on
             whose behalf SNMP messages will be generated using
             this entry."
        ::= { snmpTargetParamsEntry 4 }

    snmpTargetParamsSecurityLevel OBJECT-TYPE
        SYNTAX      SnmpSecurityLevel
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "The Level of Security to be used when generating
             SNMP messages using this entry."
        ::= { snmpTargetParamsEntry 5 }

    snmpTargetParamsStorageType OBJECT-TYPE
        SYNTAX      StorageType
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "The storage type for this conceptual row.
             Conceptual rows having the value 'permanent' need not
             allow write-access to any columnar objects in the row."
        DEFVAL { nonVolatile }
        ::= { snmpTargetParamsEntry 6 }

    snmpTargetParamsRowStatus OBJECT-TYPE
        SYNTAX      RowStatus
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "The status of this conceptual row.

             To create a row in this table, a manager must
             set this object to either createAndGo(4) or
             createAndWait(5).

             Until instances of all corresponding columns are
             appropriately configured, the value of the
             corresponding instance of the snmpTargetParamsRowStatus
             column is 'notReady'.

             In particular, a newly created row cannot be made
             active until the corresponding
             snmpTargetParamsMPModel,
             snmpTargetParamsSecurityModel,
```

```
            snmpTargetParamsSecurityName,
            and snmpTargetParamsSecurityLevel have all been set.

            The following objects may not be modified while the
            value of this object is active(1):
                - snmpTargetParamsMPModel
                - snmpTargetParamsSecurityModel
                - snmpTargetParamsSecurityName
                - snmpTargetParamsSecurityLevel
            An attempt to set these objects while the value of
            snmpTargetParamsRowStatus is active(1) will result in
            an inconsistentValue error."
        ::= { snmpTargetParamsEntry 7 }

    snmpUnavailableContexts OBJECT-TYPE
        SYNTAX       Counter32
        MAX-ACCESS   read-only
        STATUS       current
        DESCRIPTION
            "The total number of packets received by the SNMP
             engine which were dropped because the context
             contained in the message was unavailable."
        ::= { snmpTargetObjects 4 }

    snmpUnknownContexts OBJECT-TYPE
        SYNTAX       Counter32
        MAX-ACCESS   read-only
        STATUS       current
        DESCRIPTION
            "The total number of packets received by the SNMP
             engine which were dropped because the context
             contained in the message was unknown."
        ::= { snmpTargetObjects 5 }

    --
    --
    -- Conformance information
    --
    --

    snmpTargetCompliances OBJECT IDENTIFIER ::=
                                          { snmpTargetConformance 1 }
    snmpTargetGroups      OBJECT IDENTIFIER ::=
                                          { snmpTargetConformance 2 }

    --
    --
    -- Compliance statements
```

```
   --
   --

   snmpTargetCommandResponderCompliance MODULE-COMPLIANCE
       STATUS      current
       DESCRIPTION
           "The compliance statement for SNMP entities which include
            a command responder application."
       MODULE -- This Module
           MANDATORY-GROUPS { snmpTargetCommandResponderGroup }
       ::= { snmpTargetCompliances 1 }

   snmpTargetBasicGroup OBJECT-GROUP
       OBJECTS {
           snmpTargetSpinLock,
           snmpTargetAddrTDomain,
           snmpTargetAddrTAddress,
           snmpTargetAddrTagList,
           snmpTargetAddrParams,
           snmpTargetAddrStorageType,
           snmpTargetAddrRowStatus,
           snmpTargetParamsMPModel,
           snmpTargetParamsSecurityModel,
           snmpTargetParamsSecurityName,
           snmpTargetParamsSecurityLevel,
           snmpTargetParamsStorageType,
           snmpTargetParamsRowStatus
       }
       STATUS      current
       DESCRIPTION
           "A collection of objects providing basic remote
            configuration of management targets."
       ::= { snmpTargetGroups 1 }

   snmpTargetResponseGroup OBJECT-GROUP
       OBJECTS {
           snmpTargetAddrTimeout,
           snmpTargetAddrRetryCount
       }
       STATUS      current
       DESCRIPTION
           "A collection of objects providing remote configuration
            of management targets for applications which generate
            SNMP messages for which a response message would be
            expected."
       ::= { snmpTargetGroups 2 }

   snmpTargetCommandResponderGroup OBJECT-GROUP
```

```
        OBJECTS {
            snmpUnavailableContexts,
            snmpUnknownContexts
        }
        STATUS      current
        DESCRIPTION
            "A collection of objects required for command responder
             applications, used for counting error conditions."
        ::= { snmpTargetGroups 3 }

    END
```

4.2. The Notification MIB Module

The SNMP-NOTIFICATION-MIB module contains objects for the remote configuration of the parameters used by an SNMP entity for the generation of notifications. It consists of three tables and conformance/compliance statements. The first table, the snmpNotifyTable, contains entries which select which entries in the snmpTargetAddrTable should be used for generating notifications, and the type of notifications to be generated.

The second table, the snmpNotifyFilterProfileTable, sparsely augments the snmpTargetParamsTable with an object which is used to associate a set of filters with a particular management target.

The third table, the snmpNotifyFilterTable, defines filters which are used to limit the number of notifications which are generated using particular management targets.

4.2.1. Definitions

```
    SNMP-NOTIFICATION-MIB DEFINITIONS ::= BEGIN

    IMPORTS
        MODULE-IDENTITY,
        OBJECT-TYPE,
        snmpModules
            FROM SNMPv2-SMI

        RowStatus,
        StorageType
            FROM SNMPv2-TC

        SnmpAdminString
            FROM SNMP-FRAMEWORK-MIB

        SnmpTagValue,
```

```
       snmpTargetParamsName
           FROM SNMP-TARGET-MIB

       MODULE-COMPLIANCE,
       OBJECT-GROUP
           FROM SNMPv2-CONF;

   snmpNotificationMIB MODULE-IDENTITY
       LAST-UPDATED "200210140000Z"
       ORGANIZATION "IETF SNMPv3 Working Group"
       CONTACT-INFO
           "WG-email:   snmpv3@lists.tislabs.com
            Subscribe:  majordomo@lists.tislabs.com
                        In message body:  subscribe snmpv3

            Co-Chair:   Russ Mundy
                        Network Associates Laboratories
            Postal:     15204 Omega Drive, Suite 300
                        Rockville, MD 20850-4601
                        USA
            EMail:      mundy@tislabs.com
            Phone:      +1 301-947-7107

            Co-Chair:   David Harrington
                        Enterasys Networks
            Postal:     35 Industrial Way
                        P. O. Box 5004
                        Rochester, New Hampshire 03866-5005
                        USA
            EMail:      dbh@enterasys.com
            Phone:      +1 603-337-2614

            Co-editor:  David B. Levi
                        Nortel Networks
            Postal:     3505 Kesterwood Drive
                        Knoxville, Tennessee 37918
            EMail:      dlevi@nortelnetworks.com
            Phone:      +1 865 686 0432

            Co-editor:  Paul Meyer
                        Secure Computing Corporation
            Postal:     2675 Long Lake Road
                        Roseville, Minnesota 55113
            EMail:      paul_meyer@securecomputing.com
            Phone:      +1 651 628 1592

            Co-editor:  Bob Stewart
                        Retired"
```

```
        DESCRIPTION
            "This MIB module defines MIB objects which provide
             mechanisms to remotely configure the parameters
             used by an SNMP entity for the generation of
             notifications.

             Copyright (C) The Internet Society (2002). This
             version of this MIB module is part of RFC 3413;
             see the RFC itself for full legal notices.
            "
        REVISION    "200210140000Z"             -- 14 October 2002
        DESCRIPTION "Clarifications, published as
                     RFC 3413."
        REVISION    "199808040000Z"             -- 4 August 1998
        DESCRIPTION "Clarifications, published as
                     RFC 2573."
        REVISION    "199707140000Z"             -- 14 July 1997
        DESCRIPTION "The initial revision, published as RFC2273."
        ::= { snmpModules 13 }

    snmpNotifyObjects       OBJECT IDENTIFIER ::=
                                        { snmpNotificationMIB 1 }
    snmpNotifyConformance   OBJECT IDENTIFIER ::=
                                        { snmpNotificationMIB 3 }

    --
    --
    -- The snmpNotifyObjects group
    --
    --

    snmpNotifyTable OBJECT-TYPE
        SYNTAX      SEQUENCE OF SnmpNotifyEntry
        MAX-ACCESS  not-accessible
        STATUS      current
        DESCRIPTION
            "This table is used to select management targets which should
             receive notifications, as well as the type of notification
             which should be sent to each selected management target."
        ::= { snmpNotifyObjects 1 }

    snmpNotifyEntry OBJECT-TYPE
        SYNTAX      SnmpNotifyEntry
        MAX-ACCESS  not-accessible
        STATUS      current
        DESCRIPTION
            "An entry in this table selects a set of management targets
             which should receive notifications, as well as the type of
```

```
            notification which should be sent to each selected
            management target.

            Entries in the snmpNotifyTable are created and
            deleted using the snmpNotifyRowStatus object."
        INDEX { IMPLIED snmpNotifyName }
        ::= { snmpNotifyTable 1 }

    SnmpNotifyEntry ::= SEQUENCE {
        snmpNotifyName          SnmpAdminString,
        snmpNotifyTag           SnmpTagValue,
        snmpNotifyType          INTEGER,
        snmpNotifyStorageType   StorageType,
        snmpNotifyRowStatus     RowStatus
    }

    snmpNotifyName OBJECT-TYPE
        SYNTAX      SnmpAdminString (SIZE(1..32))
        MAX-ACCESS  not-accessible
        STATUS      current
        DESCRIPTION
            "The locally arbitrary, but unique identifier associated
             with this snmpNotifyEntry."
        ::= { snmpNotifyEntry 1 }

    snmpNotifyTag OBJECT-TYPE
        SYNTAX      SnmpTagValue
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "This object contains a single tag value which is used
             to select entries in the snmpTargetAddrTable.  Any entry
             in the snmpTargetAddrTable which contains a tag value
             which is equal to the value of an instance of this
             object is selected.  If this object contains a value
             of zero length, no entries are selected."
        DEFVAL { "" }
        ::= { snmpNotifyEntry 2 }

    snmpNotifyType OBJECT-TYPE
        SYNTAX      INTEGER {
                        trap(1),
                        inform(2)
                    }
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "This object determines the type of notification to
```

```
            be generated for entries in the snmpTargetAddrTable
            selected by the corresponding instance of
            snmpNotifyTag.  This value is only used when
            generating notifications, and is ignored when
            using the snmpTargetAddrTable for other purposes.

            If the value of this object is trap(1), then any
            messages generated for selected rows will contain
            Unconfirmed-Class PDUs.

            If the value of this object is inform(2), then any
            messages generated for selected rows will contain
            Confirmed-Class PDUs.

            Note that if an SNMP entity only supports
            generation of Unconfirmed-Class PDUs (and not
            Confirmed-Class PDUs), then this object may be
            read-only."
        DEFVAL { trap }
        ::= { snmpNotifyEntry 3 }

    snmpNotifyStorageType OBJECT-TYPE
        SYNTAX      StorageType
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "The storage type for this conceptual row.
             Conceptual rows having the value 'permanent' need not
             allow write-access to any columnar objects in the row."
        DEFVAL { nonVolatile }
        ::= { snmpNotifyEntry 4 }

    snmpNotifyRowStatus OBJECT-TYPE
        SYNTAX      RowStatus
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "The status of this conceptual row.

             To create a row in this table, a manager must
             set this object to either createAndGo(4) or
             createAndWait(5)."
        ::= { snmpNotifyEntry 5 }

    snmpNotifyFilterProfileTable OBJECT-TYPE
        SYNTAX      SEQUENCE OF SnmpNotifyFilterProfileEntry
        MAX-ACCESS  not-accessible
        STATUS      current
```

```
        DESCRIPTION
            "This table is used to associate a notification filter
             profile with a particular set of target parameters."
        ::= { snmpNotifyObjects 2 }

    snmpNotifyFilterProfileEntry OBJECT-TYPE
        SYNTAX      SnmpNotifyFilterProfileEntry
        MAX-ACCESS  not-accessible
        STATUS      current
        DESCRIPTION
            "An entry in this table indicates the name of the filter
             profile to be used when generating notifications using
             the corresponding entry in the snmpTargetParamsTable.

             Entries in the snmpNotifyFilterProfileTable are created
             and deleted using the snmpNotifyFilterProfileRowStatus
             object."
        INDEX { IMPLIED snmpTargetParamsName }
        ::= { snmpNotifyFilterProfileTable 1 }

    SnmpNotifyFilterProfileEntry ::= SEQUENCE {
        snmpNotifyFilterProfileName         SnmpAdminString,
        snmpNotifyFilterProfileStorType     StorageType,
        snmpNotifyFilterProfileRowStatus    RowStatus
    }

    snmpNotifyFilterProfileName OBJECT-TYPE
        SYNTAX      SnmpAdminString (SIZE(1..32))
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "The name of the filter profile to be used when generating
             notifications using the corresponding entry in the
             snmpTargetAddrTable."
        ::= { snmpNotifyFilterProfileEntry 1 }

    snmpNotifyFilterProfileStorType OBJECT-TYPE
        SYNTAX      StorageType
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "The storage type for this conceptual row.
             Conceptual rows having the value 'permanent' need not
             allow write-access to any columnar objects in the row."
        DEFVAL { nonVolatile }
        ::= { snmpNotifyFilterProfileEntry 2 }

    snmpNotifyFilterProfileRowStatus OBJECT-TYPE
```

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

             To create a row in this table, a manager must
             set this object to either createAndGo(4) or
             createAndWait(5).

             Until instances of all corresponding columns are
             appropriately configured, the value of the
             corresponding instance of the
             snmpNotifyFilterProfileRowStatus column is 'notReady'.

             In particular, a newly created row cannot be made
             active until the corresponding instance of
             snmpNotifyFilterProfileName has been set."
        ::= { snmpNotifyFilterProfileEntry 3 }

    snmpNotifyFilterTable OBJECT-TYPE
        SYNTAX      SEQUENCE OF SnmpNotifyFilterEntry
        MAX-ACCESS  not-accessible
        STATUS      current
        DESCRIPTION
            "The table of filter profiles.  Filter profiles are used
             to determine whether particular management targets should
             receive particular notifications.

             When a notification is generated, it must be compared
             with the filters associated with each management target
             which is configured to receive notifications, in order to
             determine whether it may be sent to each such management
             target.

             A more complete discussion of notification filtering
             can be found in section 6. of [SNMP-APPL]."
        ::= { snmpNotifyObjects 3 }

    snmpNotifyFilterEntry OBJECT-TYPE
        SYNTAX      SnmpNotifyFilterEntry
        MAX-ACCESS  not-accessible
        STATUS      current
        DESCRIPTION
            "An element of a filter profile.

             Entries in the snmpNotifyFilterTable are created and
             deleted using the snmpNotifyFilterRowStatus object."
```

```
        INDEX {         snmpNotifyFilterProfileName,
                  IMPLIED snmpNotifyFilterSubtree }
        ::= { snmpNotifyFilterTable 1 }

    SnmpNotifyFilterEntry ::= SEQUENCE {
        snmpNotifyFilterSubtree           OBJECT IDENTIFIER,
        snmpNotifyFilterMask              OCTET STRING,
        snmpNotifyFilterType              INTEGER,
        snmpNotifyFilterStorageType       StorageType,
        snmpNotifyFilterRowStatus         RowStatus
    }

    snmpNotifyFilterSubtree OBJECT-TYPE
        SYNTAX      OBJECT IDENTIFIER
        MAX-ACCESS  not-accessible
        STATUS      current
        DESCRIPTION
            "The MIB subtree which, when combined with the corresponding
             instance of snmpNotifyFilterMask, defines a family of
             subtrees which are included in or excluded from the
             filter profile."
        ::= { snmpNotifyFilterEntry 1 }

    snmpNotifyFilterMask OBJECT-TYPE
        SYNTAX      OCTET STRING (SIZE(0..16))
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "The bit mask which, in combination with the corresponding
             instance of snmpNotifyFilterSubtree, defines a family of
             subtrees which are included in or excluded from the
             filter profile.

             Each bit of this bit mask corresponds to a
             sub-identifier of snmpNotifyFilterSubtree, with the
             most significant bit of the i-th octet of this octet
             string value (extended if necessary, see below)
             corresponding to the (8*i - 7)-th sub-identifier, and
             the least significant bit of the i-th octet of this
             octet string corresponding to the (8*i)-th
             sub-identifier, where i is in the range 1 through 16.

             Each bit of this bit mask specifies whether or not
             the corresponding sub-identifiers must match when
             determining if an OBJECT IDENTIFIER matches this
             family of filter subtrees; a '1' indicates that an
             exact match must occur; a '0' indicates 'wild card',
             i.e., any sub-identifier value matches.
```

```
            Thus, the OBJECT IDENTIFIER X of an object instance
            is contained in a family of filter subtrees if, for
            each sub-identifier of the value of
            snmpNotifyFilterSubtree, either:

              the i-th bit of snmpNotifyFilterMask is 0, or

              the i-th sub-identifier of X is equal to the i-th
              sub-identifier of the value of
              snmpNotifyFilterSubtree.

            If the value of this bit mask is M bits long and
            there are more than M sub-identifiers in the
            corresponding instance of snmpNotifyFilterSubtree,
            then the bit mask is extended with 1's to be the
            required length.

            Note that when the value of this object is the
            zero-length string, this extension rule results in
            a mask of all-1's being used (i.e., no 'wild card'),
            and the family of filter subtrees is the one
            subtree uniquely identified by the corresponding
            instance of snmpNotifyFilterSubtree."
        DEFVAL { ''H }
        ::= { snmpNotifyFilterEntry 2 }

    snmpNotifyFilterType OBJECT-TYPE
        SYNTAX      INTEGER {
                        included(1),
                        excluded(2)
                    }
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "This object indicates whether the family of filter subtrees
             defined by this entry are included in or excluded from a
             filter.  A more detailed discussion of the use of this
             object can be found in section 6. of [SNMP-APPL]."
        DEFVAL { included }
        ::= { snmpNotifyFilterEntry 3 }

    snmpNotifyFilterStorageType OBJECT-TYPE
        SYNTAX      StorageType
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "The storage type for this conceptual row.
             Conceptual rows having the value 'permanent' need not
```

```
            allow write-access to any columnar objects in the row."
        DEFVAL { nonVolatile }
        ::= { snmpNotifyFilterEntry 4 }

    snmpNotifyFilterRowStatus OBJECT-TYPE
        SYNTAX      RowStatus
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "The status of this conceptual row.

             To create a row in this table, a manager must
             set this object to either createAndGo(4) or
             createAndWait(5)."
        ::= { snmpNotifyFilterEntry 5 }

    --
    --
    -- Conformance information
    --
    --

    snmpNotifyCompliances OBJECT IDENTIFIER ::=
                                        { snmpNotifyConformance 1 }
    snmpNotifyGroups      OBJECT IDENTIFIER ::=
                                        { snmpNotifyConformance 2 }

    --
    --
    -- Compliance statements
    --
    --

    snmpNotifyBasicCompliance MODULE-COMPLIANCE
        STATUS      current
        DESCRIPTION
            "The compliance statement for minimal SNMP entities which
             implement only SNMP Unconfirmed-Class notifications and
             read-create operations on only the snmpTargetAddrTable."
        MODULE SNMP-TARGET-MIB
            MANDATORY-GROUPS { snmpTargetBasicGroup }

            OBJECT snmpTargetParamsMPModel
            MIN-ACCESS    read-only
            DESCRIPTION
                "Create/delete/modify access is not required."

            OBJECT snmpTargetParamsSecurityModel
```

```
            MIN-ACCESS    read-only
            DESCRIPTION
                "Create/delete/modify access is not required."

            OBJECT snmpTargetParamsSecurityName
            MIN-ACCESS    read-only
            DESCRIPTION
                "Create/delete/modify access is not required."

            OBJECT snmpTargetParamsSecurityLevel
            MIN-ACCESS    read-only
            DESCRIPTION
                "Create/delete/modify access is not required."

            OBJECT snmpTargetParamsStorageType
            SYNTAX INTEGER {
                readOnly(5)
            }
            MIN-ACCESS    read-only
            DESCRIPTION
                "Create/delete/modify access is not required.
                 Support of the values other(1), volatile(2),
                 nonVolatile(3), and permanent(4) is not required."

            OBJECT snmpTargetParamsRowStatus
            SYNTAX INTEGER {
                active(1)
            }
            MIN-ACCESS    read-only
            DESCRIPTION
                "Create/delete/modify access to the
                 snmpTargetParamsTable is not required.
                 Support of the values notInService(2), notReady(3),
                 createAndGo(4), createAndWait(5), and destroy(6) is
                 not required."

        MODULE -- This Module
            MANDATORY-GROUPS { snmpNotifyGroup }

            OBJECT snmpNotifyTag
            MIN-ACCESS    read-only
            DESCRIPTION
                "Create/delete/modify access is not required."

            OBJECT snmpNotifyType
            SYNTAX INTEGER {
                trap(1)
            }
```

```
            MIN-ACCESS    read-only
            DESCRIPTION
                "Create/delete/modify access is not required.
                 Support of the value notify(2) is not required."

            OBJECT snmpNotifyStorageType
            SYNTAX INTEGER {
                readOnly(5)
            }
            MIN-ACCESS    read-only
            DESCRIPTION
                "Create/delete/modify access is not required.
                 Support of the values other(1), volatile(2),
                 nonVolatile(3), and permanent(4) is not required."

            OBJECT snmpNotifyRowStatus
            SYNTAX INTEGER {
                active(1)
            }
            MIN-ACCESS    read-only
            DESCRIPTION
                "Create/delete/modify access to the
                 snmpNotifyTable is not required.
                 Support of the values notInService(2), notReady(3),
                 createAndGo(4), createAndWait(5), and destroy(6) is
                 not required."

        ::= { snmpNotifyCompliances 1 }

    snmpNotifyBasicFiltersCompliance MODULE-COMPLIANCE
        STATUS      current
        DESCRIPTION
            "The compliance statement for SNMP entities which implement
             SNMP Unconfirmed-Class notifications with filtering, and
             read-create operations on all related tables."
        MODULE SNMP-TARGET-MIB
            MANDATORY-GROUPS { snmpTargetBasicGroup }
        MODULE -- This Module
            MANDATORY-GROUPS { snmpNotifyGroup,
                               snmpNotifyFilterGroup }
        ::= { snmpNotifyCompliances 2 }

    snmpNotifyFullCompliance MODULE-COMPLIANCE
        STATUS      current
        DESCRIPTION
            "The compliance statement for SNMP entities which either
             implement only SNMP Confirmed-Class notifications, or both
             SNMP Unconfirmed-Class and Confirmed-Class notifications,
```

```
            plus filtering and read-create operations on all related
            tables."
        MODULE SNMP-TARGET-MIB
            MANDATORY-GROUPS { snmpTargetBasicGroup,
                               snmpTargetResponseGroup }
        MODULE -- This Module
            MANDATORY-GROUPS { snmpNotifyGroup,
                               snmpNotifyFilterGroup }
        ::= { snmpNotifyCompliances 3 }

    snmpNotifyGroup OBJECT-GROUP
        OBJECTS {
            snmpNotifyTag,
            snmpNotifyType,
            snmpNotifyStorageType,
            snmpNotifyRowStatus
        }
        STATUS      current
        DESCRIPTION
            "A collection of objects for selecting which management
             targets are used for generating notifications, and the
             type of notification to be generated for each selected
             management target."
        ::= { snmpNotifyGroups 1 }

    snmpNotifyFilterGroup OBJECT-GROUP
        OBJECTS {
            snmpNotifyFilterProfileName,
            snmpNotifyFilterProfileStorType,
            snmpNotifyFilterProfileRowStatus,
            snmpNotifyFilterMask,
            snmpNotifyFilterType,
            snmpNotifyFilterStorageType,
            snmpNotifyFilterRowStatus
        }
        STATUS      current
        DESCRIPTION
            "A collection of objects providing remote configuration
             of notification filters."
        ::= { snmpNotifyGroups 2 }

    END
```

4.3. The Proxy MIB Module

The SNMP-PROXY-MIB module, which defines MIB objects that provide mechanisms to remotely configure the parameters used by an SNMP entity for proxy forwarding operations, contains a single table. This table, snmpProxyTable, is used to define translations between management targets for use when forwarding messages.

4.3.1. Definitions

```
SNMP-PROXY-MIB DEFINITIONS ::= BEGIN

IMPORTS
    MODULE-IDENTITY,
    OBJECT-TYPE,
    snmpModules
        FROM SNMPv2-SMI

    RowStatus,
    StorageType
        FROM SNMPv2-TC

    SnmpEngineID,
    SnmpAdminString
        FROM SNMP-FRAMEWORK-MIB

    SnmpTagValue
        FROM SNMP-TARGET-MIB

    MODULE-COMPLIANCE,
    OBJECT-GROUP
        FROM SNMPv2-CONF;

snmpProxyMIB MODULE-IDENTITY
    LAST-UPDATED "200210140000Z"
    ORGANIZATION "IETF SNMPv3 Working Group"
    CONTACT-INFO
        "WG-email:   snmpv3@lists.tislabs.com
         Subscribe:  majordomo@lists.tislabs.com
                     In message body:  subscribe snmpv3

         Co-Chair:   Russ Mundy
                     Network Associates Laboratories
         Postal:     15204 Omega Drive, Suite 300
                     Rockville, MD 20850-4601
                     USA
         EMail:      mundy@tislabs.com
         Phone:      +1 301-947-7107
```

```
            Co-Chair:   David Harrington
                        Enterasys Networks
            Postal:     35 Industrial Way
                        P. O. Box 5004
                        Rochester, New Hampshire 03866-5005
                        USA
            EMail:      dbh@enterasys.com
            Phone:      +1 603-337-2614

            Co-editor:  David B. Levi
                        Nortel Networks
            Postal:     3505 Kesterwood Drive
                        Knoxville, Tennessee 37918
            EMail:      dlevi@nortelnetworks.com
            Phone:      +1 865 686 0432

            Co-editor:  Paul Meyer
                        Secure Computing Corporation
            Postal:     2675 Long Lake Road
                        Roseville, Minnesota 55113
            EMail:      paul_meyer@securecomputing.com
            Phone:      +1 651 628 1592

            Co-editor:  Bob Stewart
                        Retired"
        DESCRIPTION
            "This MIB module defines MIB objects which provide
             mechanisms to remotely configure the parameters
             used by a proxy forwarding application.

             Copyright (C) The Internet Society (2002). This
             version of this MIB module is part of RFC 3413;
             see the RFC itself for full legal notices.
            "
        REVISION    "200210140000Z"             -- 14 October 2002
        DESCRIPTION "Clarifications, published as
                     RFC 3413."
        REVISION    "199808040000Z"             -- 4 August 1998
        DESCRIPTION "Clarifications, published as
                     RFC 2573."
        REVISION    "199707140000Z"             -- 14 July 1997
        DESCRIPTION "The initial revision, published as RFC2273."
        ::= { snmpModules 14 }

    snmpProxyObjects        OBJECT IDENTIFIER ::= { snmpProxyMIB 1 }
    snmpProxyConformance    OBJECT IDENTIFIER ::= { snmpProxyMIB 3 }

    --
```

```
   --
   -- The snmpProxyObjects group
   --
   --

   snmpProxyTable OBJECT-TYPE
       SYNTAX      SEQUENCE OF SnmpProxyEntry
       MAX-ACCESS  not-accessible
       STATUS      current
       DESCRIPTION
           "The table of translation parameters used by proxy forwarder
            applications for forwarding SNMP messages."
       ::= { snmpProxyObjects 2 }

   snmpProxyEntry OBJECT-TYPE
       SYNTAX      SnmpProxyEntry
       MAX-ACCESS  not-accessible
       STATUS      current
       DESCRIPTION
           "A set of translation parameters used by a proxy forwarder
            application for forwarding SNMP messages.

            Entries in the snmpProxyTable are created and deleted
            using the snmpProxyRowStatus object."
       INDEX { IMPLIED snmpProxyName }
       ::= { snmpProxyTable 1 }

   SnmpProxyEntry ::= SEQUENCE {
       snmpProxyName               SnmpAdminString,
       snmpProxyType               INTEGER,
       snmpProxyContextEngineID    SnmpEngineID,
       snmpProxyContextName        SnmpAdminString,
       snmpProxyTargetParamsIn     SnmpAdminString,
       snmpProxySingleTargetOut    SnmpAdminString,
       snmpProxyMultipleTargetOut  SnmpTagValue,
       snmpProxyStorageType        StorageType,
       snmpProxyRowStatus          RowStatus
   }

   snmpProxyName OBJECT-TYPE
       SYNTAX      SnmpAdminString (SIZE(1..32))
       MAX-ACCESS  not-accessible
       STATUS      current
       DESCRIPTION
           "The locally arbitrary, but unique identifier associated
            with this snmpProxyEntry."
       ::= { snmpProxyEntry 1 }
```

```
   snmpProxyType OBJECT-TYPE
       SYNTAX      INTEGER {
                       read(1),
                       write(2),
                       trap(3),
                       inform(4)
                   }
       MAX-ACCESS  read-create
       STATUS      current
       DESCRIPTION
           "The type of message that may be forwarded using
            the translation parameters defined by this entry."
       ::= { snmpProxyEntry 2 }

   snmpProxyContextEngineID OBJECT-TYPE
       SYNTAX      SnmpEngineID
       MAX-ACCESS  read-create
       STATUS      current
       DESCRIPTION
           "The contextEngineID contained in messages that
            may be forwarded using the translation parameters
            defined by this entry."
       ::= { snmpProxyEntry 3 }

   snmpProxyContextName OBJECT-TYPE
       SYNTAX      SnmpAdminString
       MAX-ACCESS  read-create
       STATUS      current
       DESCRIPTION
           "The contextName contained in messages that may be
            forwarded using the translation parameters defined
            by this entry.

            This object is optional, and if not supported, the
            contextName contained in a message is ignored when
            selecting an entry in the snmpProxyTable."
       ::= { snmpProxyEntry 4 }

   snmpProxyTargetParamsIn OBJECT-TYPE
       SYNTAX      SnmpAdminString
       MAX-ACCESS  read-create
       STATUS      current
       DESCRIPTION
           "This object selects an entry in the snmpTargetParamsTable.
            The selected entry is used to determine which row of the
            snmpProxyTable to use for forwarding received messages."
       ::= { snmpProxyEntry 5 }
```

```
   snmpProxySingleTargetOut OBJECT-TYPE
       SYNTAX      SnmpAdminString
       MAX-ACCESS  read-create
       STATUS      current
       DESCRIPTION
           "This object selects a management target defined in the
            snmpTargetAddrTable (in the SNMP-TARGET-MIB).  The
            selected target is defined by an entry in the
            snmpTargetAddrTable whose index value (snmpTargetAddrName)
            is equal to this object.

            This object is only used when selection of a single
            target is required (i.e. when forwarding an incoming
            read or write request)."
       ::= { snmpProxyEntry 6 }

   snmpProxyMultipleTargetOut OBJECT-TYPE
       SYNTAX      SnmpTagValue
       MAX-ACCESS  read-create
       STATUS      current
       DESCRIPTION
           "This object selects a set of management targets defined
            in the snmpTargetAddrTable (in the SNMP-TARGET-MIB).

            This object is only used when selection of multiple
            targets is required (i.e. when forwarding an incoming
            notification)."
       ::= { snmpProxyEntry 7 }

   snmpProxyStorageType OBJECT-TYPE
       SYNTAX      StorageType
       MAX-ACCESS  read-create
       STATUS      current
       DESCRIPTION
           "The storage type of this conceptual row.
            Conceptual rows having the value 'permanent' need not
            allow write-access to any columnar objects in the row."
       DEFVAL { nonVolatile }
       ::= { snmpProxyEntry 8 }

   snmpProxyRowStatus OBJECT-TYPE
       SYNTAX      RowStatus
       MAX-ACCESS  read-create
       STATUS      current
       DESCRIPTION
           "The status of this conceptual row.

            To create a row in this table, a manager must
```

```
            set this object to either createAndGo(4) or
            createAndWait(5).

            The following objects may not be modified while the
            value of this object is active(1):
                - snmpProxyType
                - snmpProxyContextEngineID
                - snmpProxyContextName
                - snmpProxyTargetParamsIn
                - snmpProxySingleTargetOut
                - snmpProxyMultipleTargetOut"
        ::= { snmpProxyEntry 9 }

    --
    --
    -- Conformance information
    --
    --

    snmpProxyCompliances OBJECT IDENTIFIER ::=
                                            { snmpProxyConformance 1 }
    snmpProxyGroups      OBJECT IDENTIFIER ::=
                                            { snmpProxyConformance 2 }

    --
    --
    -- Compliance statements
    --
    --

    snmpProxyCompliance MODULE-COMPLIANCE
        STATUS      current
        DESCRIPTION
            "The compliance statement for SNMP entities which include
             a proxy forwarding application."
        MODULE SNMP-TARGET-MIB
            MANDATORY-GROUPS { snmpTargetBasicGroup,
                               snmpTargetResponseGroup }
        MODULE -- This Module
            MANDATORY-GROUPS { snmpProxyGroup }
        ::= { snmpProxyCompliances 1 }

    snmpProxyGroup OBJECT-GROUP
        OBJECTS {
            snmpProxyType,
            snmpProxyContextEngineID,
            snmpProxyContextName,
            snmpProxyTargetParamsIn,
```

```
            snmpProxySingleTargetOut,
            snmpProxyMultipleTargetOut,
            snmpProxyStorageType,
            snmpProxyRowStatus
        }
        STATUS      current
        DESCRIPTION
            "A collection of objects providing remote configuration of
             management target translation parameters for use by
             proxy forwarder applications."
        ::= { snmpProxyGroups 3 }

    END
```

5. Identification of Management Targets in Notification Originators

This section describes the mechanisms used by a notification originator application when using the MIB module described in this document to determine the set of management targets to be used when generating a notification.

A notification originator uses all active entries in the snmpNotifyTable to find the management targets to be used for generating notifications. Each active entry in this table selects zero or more entries in the snmpTargetAddrTable. When a notification is generated, it is sent to all of the targets specified by the selected snmpTargetAddrTable entries (subject to the application of access control and notification filtering).

Any entry in the snmpTargetAddrTable whose snmpTargetAddrTagList object contains a tag value which is equal to a value of snmpNotifyTag is selected by the snmpNotifyEntry which contains that instance of snmpNotifyTag. Note that a particular snmpTargetAddrEntry may be selected by multiple entries in the snmpNotifyTable, resulting in multiple notifications being generated using that snmpTargetAddrEntry (this allows, for example, both traps and informs to be sent to the same target).

Each snmpTargetAddrEntry contains a pointer to the snmpTargetParamsTable (snmpTargetAddrParams). This pointer selects a set of SNMP parameters to be used for generating notifications. If the selected entry in the snmpTargetParamsTable does not exist, the management target is not used to generate notifications.

The decision as to whether a notification should contain an Unconfirmed-Class or a Confirmed-Class PDU is determined by the value of the snmpNotifyType object. If the value of this object is trap(1), the notification should contain an Unconfirmed-Class PDU.

If the value of this object is inform(2), then the notification should contain a Confirmed-Class PDU, and the timeout time and number of retries for the notification are the value of snmpTargetAddrTimeout and snmpTargetAddrRetryCount. Note that the exception to these rules is when the snmpTargetParamsMPModel object indicates an SNMP version which supports a different PDU version. In this case, the notification may be sent using a different PDU type ([RFC2576] defines the PDU type in the case where the outgoing SNMP version is SNMPv1).

6. Notification Filtering

This section describes the mechanisms used by a notification originator application when using the MIB module described in this document to filter generation of notifications.

A notification originator uses the snmpNotifyFilterTable to filter notifications. A notification filter profile may be associated with a particular entry in the snmpTargetParamsTable. The associated filter profile is identified by an entry in the snmpNotifyFilterProfileTable whose index is equal to the index of the entry in the snmpTargetParamsTable. If no such entry exists in the snmpNotifyFilterProfileTable, no filtering is performed for that management target.

If such an entry does exist, the value of snmpNotifyFilterProfileName of the entry is compared with the corresponding portion of the index of all active entries in the snmpNotifyFilterTable. All such entries for which this comparison results in an exact match are used for filtering a notification generated using the associated snmpTargetParamsEntry. If no such entries exist, no filtering is performed, and a notification may be sent to the management target.

Otherwise, if matching entries do exist, a notification may be sent if the NOTIFICATION-TYPE OBJECT IDENTIFIER of the notification (this is the value of the element of the variable bindings whose name is snmpTrapOID.0, i.e., the second variable binding) is specifically included, and none of the object instances to be included in the variable-bindings of the notification are specifically excluded by the matching entries.

Each set of snmpNotifyFilterTable entries is divided into two collections of filter subtrees: the included filter subtrees, and the excluded filter subtrees. The snmpNotifyFilterType object defines the collection to which each matching entry belongs.

To determine whether a particular notification name or object instance is excluded by the set of matching entries, compare the

notification name's or object instance's OBJECT IDENTIFIER with each of the matching entries. For a notification name, if none match, then the notification name is considered excluded, and the notification should not be sent to this management target. For an object instance, if none match, the object instance is considered included, and the notification may be sent to this management target. If one or more match, then the notification name or object instance is included or excluded, according to the value of snmpNotifyFilterType in the entry whose value of snmpNotifyFilterSubtree has the most sub-identifiers. If multiple entries match and have the same number of sub-identifiers, then the value of snmpNotifyFilterType, in the entry among those which match, and whose instance is lexicographically the largest, determines the inclusion or exclusion.

A notification name or object instance's OBJECT IDENTIFIER X matches an entry in the snmpNotifyFilterTable when the number of sub-identifiers in X is at least as many as in the value of snmpNotifyFilterSubtree for the entry, and each sub-identifier in the value of snmpNotifyFilterSubtree matches its corresponding sub-identifier in X. Two sub-identifiers match either if the corresponding bit of snmpNotifyFilterMask is zero (the 'wild card' value), or if the two sub-identifiers are equal.

7. Management Target Translation in Proxy Forwarder Applications

This section describes the mechanisms used by a proxy forwarder application when using the MIB module described in this document to translate incoming management target information into outgoing management target information for the purpose of forwarding messages. There are actually two mechanisms a proxy forwarder may use, one for forwarding request messages, and one for forwarding notification messages.

7.1. Management Target Translation for Request Forwarding

When forwarding request messages, the proxy forwarder will select a single entry in the snmpProxyTable. To select this entry, it will perform the following comparisons:

- The snmpProxyType must be read(1) if the request is a Read-Class PDU. The snmpProxyType must be write(2) if the request is a Write-Class PDU.

- The contextEngineID must equal the snmpProxyContextEngineID object.

- If the snmpProxyContextName object is supported, it must equal the contextName.

- The snmpProxyTargetParamsIn object identifies an entry in the snmpTargetParamsTable. The messageProcessingModel, security model, securityName, and securityLevel must match the values of snmpTargetParamsMPModel, snmpTargetParamsSecurityModel, snmpTargetParamsSecurityName, and snmpTargetParamsSecurityLevel of the identified entry in the snmpTargetParamsTable.

There may be multiple entries in the snmpProxyTable for which these comparisons succeed. The entry whose snmpProxyName has the lexicographically smallest value and for which the comparisons succeed will be selected by the proxy forwarder.

The outgoing management target information is identified by the value of the snmpProxySingleTargetOut object of the selected entry. This object identifies an entry in the snmpTargetAddrTable. The identified entry in the snmpTargetAddrTable also contains a reference to the snmpTargetParamsTable (snmpTargetAddrParams). If either the identified entry in the snmpTargetAddrTable does not exist, or the identified entry in the snmpTargetParamsTable does not exist, then this snmpProxyEntry does not identify valid forwarding information, and the proxy forwarder should attempt to identify another row.

If there is no entry in the snmpProxyTable for which all of the conditions above may be met, then there is no appropriate forwarding information, and the proxy forwarder should take appropriate actions.

Otherwise, The snmpTargetAddrTDomain, snmpTargetAddrTAddress, snmpTargetAddrTimeout, and snmpTargetRetryCount of the identified snmpTargetAddrEntry, and the snmpTargetParamsMPModel, snmpTargetParamsSecurityModel, snmpTargetParamsSecurityName, and snmpTargetParamsSecurityLevel of the identified snmpTargetParamsEntry are used as the destination management target.

7.2. Management Target Translation for Notification Forwarding

When forwarding notification messages, the proxy forwarder will select multiple entries in the snmpProxyTable. To select these entries, it will perform the following comparisons:

- The snmpProxyType must be trap(3) if the notification is an Unconfirmed-Class PDU. The snmpProxyType must be inform(4) if the request is a Confirmed-Class PDU.

- The contextEngineID must equal the snmpProxyContextEngineID object.

- If the snmpProxyContextName object is supported, it must equal the contextName.

- The snmpProxyTargetParamsIn object identifies an entry in the snmpTargetParamsTable. The messageProcessingModel, security model, securityName, and securityLevel must match the values of snmpTargetParamsMPModel, snmpTargetParamsSecurityModel, snmpTargetParamsSecurityName, and snmpTargetParamsSecurityLevel of the identified entry in the snmpTargetParamsTable.

All entries for which these conditions are met are selected. The snmpProxyMultipleTargetOut object of each such entry is used to select a set of entries in the snmpTargetAddrTable. Any snmpTargetAddrEntry whose snmpTargetAddrTagList object contains a tag value equal to the value of snmpProxyMultipleTargetOut, and whose snmpTargetAddrParams object references an existing entry in the snmpTargetParamsTable, is selected as a destination for the forwarded notification.

8. Intellectual Property

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-related documentation can be found in BCP-11. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to 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.

9. Acknowledgments

This document is the result of the efforts of the SNMPv3 Working Group. Some special thanks are in order to the following SNMPv3 WG members:

Harald Tveit Alvestrand (Maxware)
Dave Battle (SNMP Research, Inc.)
Alan Beard (Disney Worldwide Services)
Paul Berrevoets (SWI Systemware/Halcyon Inc.)

Martin Bjorklund (Ericsson)
Uri Blumenthal (IBM T.J. Watson Research Center)
Jeff Case (SNMP Research, Inc.)
John Curran (BBN)
Mike Daniele (Compaq Computer Corporation)
T. Max Devlin (Eltrax Systems)
John Flick (Hewlett Packard)
Rob Frye (MCI)
Wes Hardaker (U.C.Davis, Information Technology - D.C.A.S.)
David Harrington (Enterasys Networks)
Lauren Heintz (BMC Software, Inc.)
N.C. Hien (IBM T.J. Watson Research Center)
Michael Kirkham (InterWorking Labs, Inc.)
Dave Levi (Nortel Networks)
Louis A Mamakos (UUNET Technologies Inc.)
Joe Marzot (Nortel Networks)
Paul Meyer (Secure Computing Corporation)
Keith McCloghrie (Cisco Systems)
Bob Moore (IBM)
Russ Mundy (TIS Labs at Network Associates)
Bob Natale (ACE*COMM Corporation)
Mike O'Dell (UUNET Technologies Inc.)
Dave Perkins (DeskTalk)
Peter Polkinghorne (Brunel University)
Randy Presuhn (BMC Software, Inc.)
David Reeder (TIS Labs at Network Associates)
David Reid (SNMP Research, Inc.)
Aleksey Romanov (Quality Quorum)
Shawn Routhier (Epilogue)
Juergen Schoenwaelder (TU Braunschweig)
Bob Stewart (Cisco Systems)
Mike Thatcher (Independent Consultant)
Bert Wijnen (Lucent Technologies)

The document is based on recommendations of the IETF Security and Administrative Framework Evolution for SNMP Advisory Team. Members of that Advisory Team were:

David Harrington (Enterasys Networks)
Jeff Johnson (Cisco Systems)
David Levi (Nortel Networks)
John Linn (Openvision)
Russ Mundy (Trusted Information Systems) chair
Shawn Routhier (Epilogue)
Glenn Waters (Nortel)
Bert Wijnen (Lucent Technologies)

As recommended by the Advisory Team and the SNMPv3 Working Group Charter, the design incorporates as much as practical from previous RFCs and drafts. As a result, special thanks are due to the authors of previous designs known as SNMPv2u and SNMPv2*:

Jeff Case (SNMP Research, Inc.)
David Harrington (Enterasys Networks)
David Levi (Nortel Networks)
Keith McCloghrie (Cisco Systems)
Brian O'Keefe (Hewlett Packard)
Marshall T. Rose (Dover Beach Consulting)
Jon Saperia (BGS Systems Inc.)
Steve Waldbusser (International Network Services)
Glenn W. Waters (Bell-Northern Research Ltd.)

10. Security Considerations

The SNMP applications described in this document typically have direct access to MIB instrumentation. Thus, it is very important that these applications be strict in their application of access control as described in this document.

In addition, there may be some types of notification generator applications which, rather than accessing MIB instrumentation using access control, will obtain MIB information through other means (such as from a command line). The implementors and users of such applications must be responsible for not divulging MIB information that normally would be inaccessible due to access control.

Finally, the MIBs described in this document contain potentially sensitive information. A security administrator may wish to limit access to these MIBs.

11. References

11.1 Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2578] McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M. and S. Waldbusser, "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, April 1999.

[RFC2579] McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M. and S. Waldbusser, "Textual Conventions for SMIv2", STD 58, RFC 2579, April 1999.

[RFC2580] McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M. and S. Waldbusser, "Conformance Statements for SMIv2", STD 58, RFC 2580, April 1999.

[RFC3411] Harrington, D., Presuhn, R. and B. Wijnen, "An Architecture for describing Simple Network Management Protocol (SNMP) Management Frameworks", STD 62, RFC 3411, December 2002.

[RFC3412] 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.

[RFC3415] 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.

[RFC3416] Presuhn, R., Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Protocol Operations for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3416, December 2002.

[RFC3418] Presuhn, R., Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Management Information Base (MIB) for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3418, December 2002.

11.2 Informative References

[RFC1157] Case, J., Fedor, M., Schoffstall, M. and J. Davin, "Simple Network Management Protocol", STD 15, RFC 1157, May 1990.

[RFC1213] McCloghrie, K. and M. Rose, Editors, "Management Information Base for Network Management of TCP/IP-based internets: MIB-II", STD 17, RFC 1213, March 1991.

[RFC2576] Frye, R.,Levi, D., Routhier, S. and B. Wijnen, "Coexistence between Version 1, Version 2, and Version 3 of the Internet-standard Network Management Framework", RFC 2576, February 1999.

Appendix A - Trap Configuration Example

This section describes an example configuration for a Notification Generator application which implements the snmpNotifyBasicCompliance level. The example configuration specifies that the Notification Generator should send notifications to 3 separate managers, using authentication and no privacy for the first 2 managers, and using both authentication and privacy for the third manager.

The configuration consists of three rows in the snmpTargetAddrTable, two rows in the snmpTargetTable, and two rows in the snmpNotifyTable.

```
   * snmpTargetAddrName        = "addr1"
     snmpTargetAddrTDomain     = snmpUDPDomain
     snmpTargetAddrTAddress    = 128.1.2.3/162
     snmpTargetAddrTagList     = "group1"
     snmpTargetAddrParams      = "AuthNoPriv-joe"
     snmpTargetAddrStorageType = readOnly(5)
     snmpTargetAddrRowStatus   = active(1)

   * snmpTargetAddrName        = "addr2"
     snmpTargetAddrTDomain     = snmpUDPDomain
     snmpTargetAddrTAddress    = 128.2.4.6/162
     snmpTargetAddrTagList     = "group1"
     snmpTargetAddrParams      = "AuthNoPriv-joe"
     snmpTargetAddrStorageType = readOnly(5)
     snmpTargetAddrRowStatus   = active(1)

   * snmpTargetAddrName        = "addr3"
     snmpTargetAddrTDomain     = snmpUDPDomain
     snmpTargetAddrTAddress    = 128.1.5.9/162
     snmpTargetAddrTagList     = "group2"
     snmpTargetAddrParams      = "AuthPriv-bob"
     snmpTargetAddrStorageType = readOnly(5)
     snmpTargetAddrRowStatus   = active(1)

   * snmpTargetParamsName                  = "AuthNoPriv-joe"
     snmpTargetParamsMPModel               = 3
     snmpTargetParamsSecurityModel         = 3 (USM)
     snmpTargetParamsSecurityName          = "joe"
     snmpTargetParamsSecurityLevel         = authNoPriv(2)
     snmpTargetParamsStorageType           = readOnly(5)
     snmpTargetParamsRowStatus             = active(1)
```

```
    * snmpTargetParamsName                  = "AuthPriv-bob"
      snmpTargetParamsMPModel               = 3
      snmpTargetParamsSecurityModel         = 3 (USM)
      snmpTargetParamsSecurityName          = "bob"
      snmpTargetParamsSecurityLevel         = authPriv(3)
      snmpTargetParamsStorageType           = readOnly(5)
      snmpTargetParamsRowStatus             = active(1)

    * snmpNotifyName         = "group1"
      snmpNotifyTag          = "group1"
      snmpNotifyType         = trap(1)
      snmpNotifyStorageType  = readOnly(5)
      snmpNotifyRowStatus    = active(1)

    * snmpNotifyName         = "group2"
      snmpNotifyTag          = "group2"
      snmpNotifyType         = trap(1)
      snmpNotifyStorageType  = readOnly(5)
      snmpNotifyRowStatus    = active(1)
```

These entries define two groups of management targets. The first group contains two management targets:

	first target	second target
messageProcessingModel	SNMPv3	SNMPv3
securityModel	3 (USM)	3 (USM)
securityName	"joe"	"joe"
securityLevel	authNoPriv(2)	authNoPriv(2)
transportDomain	snmpUDPDomain	snmpUDPDomain
transportAddress	128.1.2.3/162	128.2.4.6/162

And the second group contains a single management target:

messageProcessingModel	SNMPv3
securityLevel	authPriv(3)
securityModel	3 (USM)
securityName	"bob"
transportDomain	snmpUDPDomain
transportAddress	128.1.5.9/162

Editors' Addresses

David B. Levi
Nortel Networks
3505 Kesterwood Drive
Knoxville, TN 37918
U.S.A.

Phone: +1 865 686 0432
EMail: dlevi@nortelnetworks.com

Paul Meyer
Secure Computing Corporation
2675 Long Lake Road
Roseville, MN 55113
U.S.A.

Phone: +1 651 628 1592
EMail: paul_meyer@securecomputing.com

Bob Stewart
Retired

Full Copyright Statement

Acknowledgement

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

Network Working Group
Request for Comments: 3414
STD: 62
Obsoletes: 2574
Category: Standards Track

U. Blumenthal
B. Wijnen
Lucent Technologies

December 2002

User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)

Status of this Memo

Copyright Notice

Abstract

This document describes the User-based Security Model (USM) for Simple Network Management Protocol (SNMP) version 3 for use in the SNMP architecture. It defines the Elements of Procedure for providing SNMP message level security. This document also includes a Management Information Base (MIB) for remotely monitoring/managing the configuration parameters for this Security Model. This document obsoletes RFC 2574.

Table of Contents

1. Introduction

The Architecture for describing Internet Management Frameworks [RFC3411] describes that an SNMP engine is composed of:

1) a Dispatcher,
2) a Message Processing Subsystem,
3) a Security Subsystem, and
4) an Access Control Subsystem.

Applications make use of the services of these subsystems.

It is important to understand the SNMP architecture and the terminology of the architecture to understand where the Security Model described in this document fits into the architecture and interacts with other subsystems within the architecture. The reader is expected to have read and understood the description of the SNMP architecture, as defined in [RFC3411].

This memo describes the User-based Security Model as it is used within the SNMP Architecture. The main idea is that we use the traditional concept of a user (identified by a userName) with which to associate security information.

This memo describes the use of HMAC-MD5-96 and HMAC-SHA-96 as the authentication protocols and the use of CBC-DES as the privacy protocol. The User-based Security Model however allows for other such protocols to be used instead of or concurrent with these protocols. Therefore, the description of HMAC-MD5-96, HMAC-SHA-96 and CBC-DES are in separate sections to reflect their self-contained nature and to indicate that they can be replaced or supplemented in the future.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

1.1. Threats

Several of the classical threats to network protocols are applicable to the network management problem and therefore would be applicable to any SNMP Security Model. Other threats are not applicable to the network management problem. This section discusses principal threats, secondary threats, and threats which are of lesser importance.

The principal threats against which this SNMP Security Model should provide protection are:

- Modification of Information The modification threat is the danger that some unauthorized entity may alter in-transit SNMP messages generated on behalf of an authorized principal in such a way as to effect unauthorized management operations, including falsifying the value of an object.

- Masquerade The masquerade threat is the danger that management operations not authorized for some user may be attempted by assuming the identity of another user that has the appropriate authorizations.

Two secondary threats are also identified. The Security Model defined in this memo provides limited protection against:

- Disclosure The disclosure threat is the danger of eavesdropping on the exchanges between managed agents and a management station. Protecting against this threat may be required as a matter of local policy.

- Message Stream Modification The SNMP protocol is typically based upon a connection-less transport service which may operate over any sub-network service. The re-ordering, delay or replay of messages can and does occur through the natural operation of many such sub-network services. The message stream modification threat is the danger that messages may be maliciously re-ordered, delayed or replayed to an extent which is greater than can occur through the natural operation of a sub-network service, in order to effect unauthorized management operations.

There are at least two threats that an SNMP Security Model need not protect against. The security protocols defined in this memo do not provide protection against:

- Denial of Service This SNMP Security Model does not attempt to address the broad range of attacks by which service on behalf of authorized users is denied. Indeed, such denial-of-service attacks are in many cases indistinguishable from the type of network failures with which any viable network management protocol must cope as a matter of course.

- Traffic Analysis This SNMP Security Model does not attempt to address traffic analysis attacks. Indeed, many traffic patterns are predictable - devices may be managed on a regular basis by a relatively small number of management applications - and therefore there is no significant advantage afforded by protecting against traffic analysis.

1.2. Goals and Constraints

Based on the foregoing account of threats in the SNMP network management environment, the goals of this SNMP Security Model are as follows.

1) Provide for verification that each received SNMP message has not been modified during its transmission through the network.

2) Provide for verification of the identity of the user on whose behalf a received SNMP message claims to have been generated.

3) Provide for detection of received SNMP messages, which request or contain management information, whose time of generation was not recent.

4) Provide, when necessary, that the contents of each received SNMP message are protected from disclosure.

In addition to the principal goal of supporting secure network management, the design of this SNMP Security Model is also influenced by the following constraints:

1) When the requirements of effective management in times of network stress are inconsistent with those of security, the design of USM has given preference to the former.

2) Neither the security protocol nor its underlying security mechanisms should depend upon the ready availability of other network services (e.g., Network Time Protocol (NTP) or key management protocols).

3) A security mechanism should entail no changes to the basic SNMP network management philosophy.

1.3. Security Services

The security services necessary to support the goals of this SNMP Security Model are as follows:

- Data Integrity is the provision of the property that data has not been altered or destroyed in an unauthorized manner, nor have data sequences been altered to an extent greater than can occur non-maliciously.

- Data Origin Authentication is the provision of the property that the claimed identity of the user on whose behalf received data was originated is corroborated.

- Data Confidentiality is the provision of the property that information is not made available or disclosed to unauthorized individuals, entities, or processes.

- Message timeliness and limited replay protection is the provision of the property that a message whose generation time is outside of a specified time window is not accepted. Note that message reordering is not dealt with and can occur in normal conditions too.

For the protocols specified in this memo, it is not possible to assure the specific originator of a received SNMP message; rather, it is the user on whose behalf the message was originated that is authenticated.

For these protocols, it not possible to obtain data integrity without data origin authentication, nor is it possible to obtain data origin authentication without data integrity. Further, there is no provision for data confidentiality without both data integrity and data origin authentication.

The security protocols used in this memo are considered acceptably secure at the time of writing. However, the procedures allow for new authentication and privacy methods to be specified at a future time if the need arises.

1.4. Module Organization

The security protocols defined in this memo are split in three different modules and each has its specific responsibilities such that together they realize the goals and security services described above:

- The authentication module MUST provide for:
  - Data Integrity,
  - Data Origin Authentication,
- The timeliness module MUST provide for:
  - Protection against message delay or replay (to an extent greater than can occur through normal operation).
- The privacy module MUST provide for
  - Protection against disclosure of the message payload.

The timeliness module is fixed for the User-based Security Model while there is provision for multiple authentication and/or privacy modules, each of which implements a specific authentication or privacy protocol respectively.

1.4.1. Timeliness Module

Section 3 (Elements of Procedure) uses the timeliness values in an SNMP message to do timeliness checking. The timeliness check is only performed if authentication is applied to the message. Since the complete message is checked for integrity, we can assume that the timeliness values in a message that passes the authentication module are trustworthy.

1.4.2. Authentication Protocol

Section 6 describes the HMAC-MD5-96 authentication protocol which is the first authentication protocol that MUST be supported with the User-based Security Model. Section 7 describes the HMAC-SHA-96 authentication protocol which is another authentication protocol that SHOULD be supported with the User-based Security Model. In the future additional or replacement authentication protocols may be defined as new needs arise.

The User-based Security Model prescribes that, if authentication is used, then the complete message is checked for integrity in the authentication module.

For a message to be authenticated, it needs to pass authentication check by the authentication module and the timeliness check which is a fixed part of this User-based Security model.

1.4.3. Privacy Protocol

Section 8 describes the CBC-DES Symmetric Encryption Protocol which is the first privacy protocol to be used with the User-based Security Model. In the future additional or replacement privacy protocols may be defined as new needs arise.

The User-based Security Model prescribes that the scopedPDU is protected from disclosure when a message is sent with privacy.

The User-based Security Model also prescribes that a message needs to be authenticated if privacy is in use.

1.5. Protection against Message Replay, Delay and Redirection

1.5.1. Authoritative SNMP Engine

In order to protect against message replay, delay and redirection, one of the SNMP engines involved in each communication is designated to be the authoritative SNMP engine. When an SNMP message contains a payload which expects a response (those messages that contain a Confirmed Class PDU [RFC3411]), then the receiver of such messages is authoritative. When an SNMP message contains a payload which does not expect a response (those messages that contain an Unconfirmed Class PDU [RFC3411]), then the sender of such a message is authoritative.

1.5.2. Mechanisms

The following mechanisms are used:

1) To protect against the threat of message delay or replay (to an extent greater than can occur through normal operation), a set of timeliness indicators (for the authoritative SNMP engine) are included in each message generated. An SNMP engine evaluates the timeliness indicators to determine if a received message is recent. An SNMP engine may evaluate the timeliness indicators to ensure that a received message is at least as recent as the last message it received from the same source. A non-authoritative SNMP engine uses received authentic messages to advance its notion of the timeliness indicators at the remote authoritative source.

   An SNMP engine MUST also use a mechanism to match incoming Responses to outstanding Requests and it MUST drop any Responses that do not match an outstanding request. For example, a msgID can be inserted in every message to cater for this functionality.

   These mechanisms provide for the detection of authenticated messages whose time of generation was not recent.

   This protection against the threat of message delay or replay does not imply nor provide any protection against unauthorized deletion or suppression of messages. Also, an SNMP engine may not be able to detect message reordering if all the messages involved are sent within the Time Window interval. Other mechanisms defined independently of the security protocol can also be used to detect the re-ordering replay, deletion, or suppression of messages containing Set operations (e.g., the MIB variable snmpSetSerialNo [RFC3418]).

2) Verification that a message sent to/from one authoritative SNMP engine cannot be replayed to/as-if-from another authoritative SNMP engine.

   Included in each message is an identifier unique to the authoritative SNMP engine associated with the sender or intended recipient of the message.

   A message containing an Unconfirmed Class PDU sent by an authoritative SNMP engine to one non-authoritative SNMP engine can potentially be replayed to another non-authoritative SNMP engine. The latter non-authoritative SNMP engine might (if it knows about the same userName with the same secrets at the authoritative SNMP engine) as a result update its notion of timeliness indicators of the authoritative SNMP engine, but that is not considered a threat. In this case, A Report or Response message will be discarded by the Message Processing Model, because there should not be an outstanding Request message. A Trap will possibly be accepted. Again, that is not considered a threat, because the communication was authenticated and timely. It is as if the authoritative SNMP engine was configured to start sending Traps to the second SNMP engine, which theoretically can happen without the knowledge of the second SNMP engine anyway. Anyway, the second SNMP engine may not expect to receive this Trap, but is allowed to see the management information contained in it.

3) Detection of messages which were not recently generated.

   A set of time indicators are included in the message, indicating the time of generation. Messages without recent time indicators are not considered authentic. In addition, an SNMP engine MUST drop any Responses that do not match an outstanding request. This however is the responsibility of the Message Processing Model.

This memo allows the same user to be defined on multiple SNMP engines. Each SNMP engine maintains a value, snmpEngineID, which uniquely identifies the SNMP engine. This value is included in each message sent to/from the SNMP engine that is authoritative (see section 1.5.1). On receipt of a message, an authoritative SNMP engine checks the value to ensure that it is the intended recipient, and a non-authoritative SNMP engine uses the value to ensure that the message is processed using the correct state information.

Each SNMP engine maintains two values, snmpEngineBoots and snmpEngineTime, which taken together provide an indication of time at that SNMP engine. Both of these values are included in an authenticated message sent to/received from that SNMP engine. On receipt, the values are checked to ensure that the indicated

timeliness value is within a Time Window of the current time. The Time Window represents an administrative upper bound on acceptable delivery delay for protocol messages.

For an SNMP engine to generate a message which an authoritative SNMP engine will accept as authentic, and to verify that a message received from that authoritative SNMP engine is authentic, such an SNMP engine must first achieve timeliness synchronization with the authoritative SNMP engine. See section 2.3.

1.6. Abstract Service Interfaces

Abstract service interfaces have been defined to describe the conceptual interfaces between the various subsystems within an SNMP entity. Similarly a set of abstract service interfaces have been defined within the User-based Security Model (USM) to describe the conceptual interfaces between the generic USM services and the self-contained authentication and privacy services.

These abstract service interfaces are defined by a set of primitives that define the services provided and the abstract data elements that must be passed when the services are invoked. This section lists the primitives that have been defined for the User-based Security Model.

1.6.1. User-based Security Model Primitives for Authentication

The User-based Security Model provides the following internal primitives to pass data back and forth between the Security Model itself and the authentication service:

```
statusInformation =
  authenticateOutgoingMsg(
  IN   authKey                  -- secret key for authentication
  IN   wholeMsg                 -- unauthenticated complete message
  OUT  authenticatedWholeMsg    -- complete authenticated message
       )

statusInformation =
  authenticateIncomingMsg(
  IN   authKey                  -- secret key for authentication
  IN   authParameters           -- as received on the wire
  IN   wholeMsg                 -- as received on the wire
  OUT  authenticatedWholeMsg    -- complete authenticated message
       )
```

1.6.2. User-based Security Model Primitives for Privacy

The User-based Security Model provides the following internal primitives to pass data back and forth between the Security Model itself and the privacy service:

```
statusInformation =
  encryptData(
  IN    encryptKey              -- secret key for encryption
  IN    dataToEncrypt           -- data to encrypt (scopedPDU)
  OUT   encryptedData           -- encrypted data (encryptedPDU)
  OUT   privParameters          -- filled in by service provider
        )

statusInformation =
  decryptData(
  IN    decryptKey              -- secret key for decrypting
  IN    privParameters          -- as received on the wire
  IN    encryptedData           -- encrypted data (encryptedPDU)
  OUT   decryptedData           -- decrypted data (scopedPDU)
        )
```

2. Elements of the Model

This section contains definitions required to realize the security model defined by this memo.

2.1. User-based Security Model Users

Management operations using this Security Model make use of a defined set of user identities. For any user on whose behalf management operations are authorized at a particular SNMP engine, that SNMP engine must have knowledge of that user. An SNMP engine that wishes to communicate with another SNMP engine must also have knowledge of a user known to that engine, including knowledge of the applicable attributes of that user.

A user and its attributes are defined as follows:

userName
: A string representing the name of the user.

securityName
: A human-readable string representing the user in a format that is Security Model independent. There is a one-to-one relationship between userName and securityName.

authProtocol
An indication of whether messages sent on behalf of this user can be authenticated, and if so, the type of authentication protocol which is used. Two such protocols are defined in this memo:

- the HMAC-MD5-96 authentication protocol.
- the HMAC-SHA-96 authentication protocol.

authKey
If messages sent on behalf of this user can be authenticated, the (private) authentication key for use with the authentication protocol. Note that a user's authentication key will normally be different at different authoritative SNMP engines. The authKey is not accessible via SNMP. The length requirements of the authKey are defined by the authProtocol in use.

authKeyChange and authOwnKeyChange
The only way to remotely update the authentication key. Does that in a secure manner, so that the update can be completed without the need to employ privacy protection.

privProtocol
An indication of whether messages sent on behalf of this user can be protected from disclosure, and if so, the type of privacy protocol which is used. One such protocol is defined in this memo: the CBC-DES Symmetric Encryption Protocol.

privKey
If messages sent on behalf of this user can be en/decrypted, the (private) privacy key for use with the privacy protocol. Note that a user's privacy key will normally be different at different authoritative SNMP engines. The privKey is not accessible via SNMP. The length requirements of the privKey are defined by the privProtocol in use.

privKeyChange and privOwnKeyChange
The only way to remotely update the encryption key. Does that in a secure manner, so that the update can be completed without the need to employ privacy protection.

2.2. Replay Protection

Each SNMP engine maintains three objects:

- snmpEngineID, which (at least within an administrative domain) uniquely and unambiguously identifies an SNMP engine.

- snmpEngineBoots, which is a count of the number of times the SNMP engine has re-booted/re-initialized since snmpEngineID was last configured; and,

- snmpEngineTime, which is the number of seconds since the snmpEngineBoots counter was last incremented.

Each SNMP engine is always authoritative with respect to these objects in its own SNMP entity. It is the responsibility of a non-authoritative SNMP engine to synchronize with the authoritative SNMP engine, as appropriate.

An authoritative SNMP engine is required to maintain the values of its snmpEngineID and snmpEngineBoots in non-volatile storage.

2.2.1. msgAuthoritativeEngineID

The msgAuthoritativeEngineID value contained in an authenticated message is used to defeat attacks in which messages from one SNMP engine to another SNMP engine are replayed to a different SNMP engine. It represents the snmpEngineID at the authoritative SNMP engine involved in the exchange of the message.

When an authoritative SNMP engine is first installed, it sets its local value of snmpEngineID according to a enterprise-specific algorithm (see the definition of the Textual Convention for SnmpEngineID in the SNMP Architecture document [RFC3411]).

2.2.2. msgAuthoritativeEngineBoots and msgAuthoritativeEngineTime

The msgAuthoritativeEngineBoots and msgAuthoritativeEngineTime values contained in an authenticated message are used to defeat attacks in which messages are replayed when they are no longer valid. They represent the snmpEngineBoots and snmpEngineTime values at the authoritative SNMP engine involved in the exchange of the message.

Through use of snmpEngineBoots and snmpEngineTime, there is no requirement for an SNMP engine to have a non-volatile clock which ticks (i.e., increases with the passage of time) even when the SNMP engine is powered off. Rather, each time an SNMP engine re-boots, it retrieves, increments, and then stores snmpEngineBoots in non-volatile storage, and resets snmpEngineTime to zero.

When an SNMP engine is first installed, it sets its local values of snmpEngineBoots and snmpEngineTime to zero. If snmpEngineTime ever reaches its maximum value (2147483647), then snmpEngineBoots is incremented as if the SNMP engine has re-booted and snmpEngineTime is reset to zero and starts incrementing again.

Each time an authoritative SNMP engine re-boots, any SNMP engines holding that authoritative SNMP engine's values of snmpEngineBoots and snmpEngineTime need to re-synchronize prior to sending correctly authenticated messages to that authoritative SNMP engine (see Section 2.3 for (re-)synchronization procedures). Note, however, that the procedures do provide for a notification to be accepted as authentic by a receiving SNMP engine, when sent by an authoritative SNMP engine which has re-booted since the receiving SNMP engine last (re-)synchronized.

If an authoritative SNMP engine is ever unable to determine its latest snmpEngineBoots value, then it must set its snmpEngineBoots value to 2147483647.

Whenever the local value of snmpEngineBoots has the value 2147483647 it latches at that value and an authenticated message always causes an notInTimeWindow authentication failure.

In order to reset an SNMP engine whose snmpEngineBoots value has reached the value 2147483647, manual intervention is required. The engine must be physically visited and re-configured, either with a new snmpEngineID value, or with new secret values for the authentication and privacy protocols of all users known to that SNMP engine. Note that even if an SNMP engine re-boots once a second that it would still take approximately 68 years before the max value of 2147483647 would be reached.

2.2.3. Time Window

The Time Window is a value that specifies the window of time in which a message generated on behalf of any user is valid. This memo specifies that the same value of the Time Window, 150 seconds, is used for all users.

2.3. Time Synchronization

Time synchronization, required by a non-authoritative SNMP engine in order to proceed with authentic communications, has occurred when the non-authoritative SNMP engine has obtained a local notion of the authoritative SNMP engine's values of snmpEngineBoots and snmpEngineTime from the authoritative SNMP engine. These values must be (and remain) within the authoritative SNMP engine's Time Window. So the local notion of the authoritative SNMP engine's values must be kept loosely synchronized with the values stored at the authoritative SNMP engine. In addition to keeping a local copy of snmpEngineBoots and snmpEngineTime from the authoritative SNMP engine, a non-authoritative SNMP engine must also keep one

local variable, latestReceivedEngineTime. This value records the highest value of snmpEngineTime that was received by the non-authoritative SNMP engine from the authoritative SNMP engine and is used to eliminate the possibility of replaying messages that would prevent the non-authoritative SNMP engine's notion of the snmpEngineTime from advancing.

A non-authoritative SNMP engine must keep local notions of these values (snmpEngineBoots, snmpEngineTime and latestReceivedEngineTime) for each authoritative SNMP engine with which it wishes to communicate. Since each authoritative SNMP engine is uniquely and unambiguously identified by its value of snmpEngineID, the non-authoritative SNMP engine may use this value as a key in order to cache its local notions of these values.

Time synchronization occurs as part of the procedures of receiving an SNMP message (Section 3.2, step 7b). As such, no explicit time synchronization procedure is required by a non-authoritative SNMP engine. Note, that whenever the local value of snmpEngineID is changed (e.g., through discovery) or when secure communications are first established with an authoritative SNMP engine, the local values of snmpEngineBoots and latestReceivedEngineTime should be set to zero. This will cause the time synchronization to occur when the next authentic message is received.

2.4. SNMP Messages Using this Security Model

The syntax of an SNMP message using this Security Model adheres to the message format defined in the version-specific Message Processing Model document (for example [RFC3412]).

The field msgSecurityParameters in SNMPv3 messages has a data type of OCTET STRING. Its value is the BER serialization of the following ASN.1 sequence:

```
USMSecurityParametersSyntax DEFINITIONS IMPLICIT TAGS ::= BEGIN

   UsmSecurityParameters ::=
       SEQUENCE {
        -- global User-based security parameters
           msgAuthoritativeEngineID     OCTET STRING,
           msgAuthoritativeEngineBoots  INTEGER (0..2147483647),
           msgAuthoritativeEngineTime   INTEGER (0..2147483647),
           msgUserName                  OCTET STRING (SIZE(0..32)),
        -- authentication protocol specific parameters
           msgAuthenticationParameters  OCTET STRING,
        -- privacy protocol specific parameters
           msgPrivacyParameters         OCTET STRING
```

```
        }
END
```

The fields of this sequence are:

- The msgAuthoritativeEngineID specifies the snmpEngineID of the authoritative SNMP engine involved in the exchange of the message.

- The msgAuthoritativeEngineBoots specifies the snmpEngineBoots value at the authoritative SNMP engine involved in the exchange of the message.

- The msgAuthoritativeEngineTime specifies the snmpEngineTime value at the authoritative SNMP engine involved in the exchange of the message.

- The msgUserName specifies the user (principal) on whose behalf the message is being exchanged. Note that a zero-length userName will not match any user, but it can be used for snmpEngineID discovery.

- The msgAuthenticationParameters are defined by the authentication protocol in use for the message, as defined by the usmUserAuthProtocol column in the user's entry in the usmUserTable.

- The msgPrivacyParameters are defined by the privacy protocol in use for the message, as defined by the usmUserPrivProtocol column in the user's entry in the usmUserTable).

See appendix A.4 for an example of the BER encoding of field msgSecurityParameters.

2.5. Services provided by the User-based Security Model

This section describes the services provided by the User-based Security Model with their inputs and outputs.

The services are described as primitives of an abstract service interface and the inputs and outputs are described as abstract data elements as they are passed in these abstract service primitives.

2.5.1. Services for Generating an Outgoing SNMP Message

When the Message Processing (MP) Subsystem invokes the User-based Security module to secure an outgoing SNMP message, it must use the appropriate service as provided by the Security module. These two services are provided:

1) A service to generate a Request message. The abstract service primitive is:

```
statusInformation =            -- success or errorIndication
  generateRequestMsg(
  IN   messageProcessingModel  -- typically, SNMP version
  IN   globalData              -- message header, admin data
  IN   maxMessageSize          -- of the sending SNMP entity
  IN   securityModel           -- for the outgoing message
  IN   securityEngineID        -- authoritative SNMP entity
  IN   securityName            -- on behalf of this principal
  IN   securityLevel           -- Level of Security requested
  IN   scopedPDU               -- message (plaintext) payload
  OUT  securityParameters      -- filled in by Security Module
  OUT  wholeMsg                -- complete generated message
  OUT  wholeMsgLength          -- length of generated message
       )
```

2) A service to generate a Response message. The abstract service primitive is:

```
statusInformation =            -- success or errorIndication
  generateResponseMsg(
  IN   messageProcessingModel  -- typically, SNMP version
  IN   globalData              -- message header, admin data
  IN   maxMessageSize          -- of the sending SNMP entity
  IN   securityModel           -- for the outgoing message
  IN   securityEngineID        -- authoritative SNMP entity
  IN   securityName            -- on behalf of this principal
  IN   securityLevel           -- Level of Security requested
  IN   scopedPDU               -- message (plaintext) payload
  IN   securityStateReference  -- reference to security state
                               -- information from original
                               -- request
  OUT  securityParameters      -- filled in by Security Module
  OUT  wholeMsg                -- complete generated message
  OUT  wholeMsgLength          -- length of generated message
       )
```

The abstract data elements passed as parameters in the abstract service primitives are as follows:

statusInformation
An indication of whether the encoding and securing of the message was successful. If not it is an indication of the problem.

messageProcessingModel
The SNMP version number for the message to be generated. This data is not used by the User-based Security module.

globalData
The message header (i.e., its administrative information). This data is not used by the User-based Security module.

maxMessageSize
The maximum message size as included in the message. This data is not used by the User-based Security module.

securityParameters
These are the security parameters. They will be filled in by the User-based Security module.

securityModel
The securityModel in use. Should be User-based Security Model. This data is not used by the User-based Security module.

securityName
Together with the snmpEngineID it identifies a row in the usmUserTablethat is to be used for securing the message. The securityName has a format that is independent of the Security Model. In case of a response this parameter is ignored and the value from the cache is used.

securityLevel
The Level of Security from which the User-based Security module determines if the message needs to be protected from disclosure and if the message needs to be authenticated.

securityEngineID
The snmpEngineID of the authoritative SNMP engine to which a dateRequest message is to be sent. In case of a response it is implied to be the processing SNMP engine’s snmpEngineID and so if it is specified, then it is ignored.

scopedPDU
The message payload. The data is opaque as far as the User-based Security Model is concerned.

securityStateReference
A handle/reference to cachedSecurityData to be used when securing an outgoing Response message. This is the exact same handle/reference as it was generated by the User-based Security module when processing the incoming Request message to which this is the Response message.

wholeMsg
The fully encoded and secured message ready for sending on the wire.

wholeMsgLength
The length of the encoded and secured message (wholeMsg).

Upon completion of the process, the User-based Security module returns statusInformation. If the process was successful, the completed message with privacy and authentication applied if such was requested by the specified securityLevel is returned. If the process was not successful, then an errorIndication is returned.

2.5.2. Services for Processing an Incoming SNMP Message

When the Message Processing (MP) Subsystem invokes the User-based Security module to verify proper security of an incoming message, it must use the service provided for an incoming message. The abstract service primitive is:

```
statusInformation =             -- errorIndication or success
                                -- error counter OID/value if error
  processIncomingMsg(
  IN   messageProcessingModel   -- typically, SNMP version
  IN   maxMessageSize           -- of the sending SNMP entity
  IN   securityParameters       -- for the received message
  IN   securityModel            -- for the received message
  IN   securityLevel            -- Level of Security
  IN   wholeMsg                 -- as received on the wire
  IN   wholeMsgLength           -- length as received on the wire
  OUT  securityEngineID         -- authoritative SNMP entity
  OUT  securityName             -- identification of the principal
  OUT  scopedPDU,               -- message (plaintext) payload
  OUT  maxSizeResponseScopedPDU -- maximum size of the Response PDU
  OUT  securityStateReference   -- reference to security state
       )                        -- information, needed for response
```

The abstract data elements passed as parameters in the abstract service primitives are as follows:

statusInformation
An indication of whether the process was successful or not. If not, then the statusInformation includes the OID and the value of the error counter that was incremented.

messageProcessingModel
The SNMP version number as received in the message. This data is not used by the User-based Security module.

maxMessageSize
The maximum message size as included in the message. The User-bas User-based Security module uses this value to calculate the maxSizeResponseScopedPDU.

securityParameters
These are the security parameters as received in the message.

securityModel
The securityModel in use. Should be the User-based Security Model. This data is not used by the User-based Security module.

securityLevel
The Level of Security from which the User-based Security module determines if the message needs to be protected from disclosure and if the message needs to be authenticated.

wholeMsg
The whole message as it was received.

wholeMsgLength
The length of the message as it was received (wholeMsg).

securityEngineID
The snmpEngineID that was extracted from the field msgAuthoritativeEngineID and that was used to lookup the secrets in the usmUserTable.

securityName
The security name representing the user on whose behalf the message was received. The securityName has a format that is independent of the Security Model.

scopedPDU
The message payload. The data is opaque as far as the User-based Security Model is concerned.

maxSizeResponseScopedPDU
The maximum size of a scopedPDU to be included in a possible Response message. The User-based Security module calculates this size based on the msgMaxSize (as received in the message) and the space required for the message header (including the securityParameters) for such a Response message.

securityStateReference
A handle/reference to cachedSecurityData to be used when securing an outgoing Response message. When the Message Processing Subsystem calls the User-based Security module to generate a

response to this incoming message it must pass this handle/reference.

Upon completion of the process, the User-based Security module returns statusInformation and, if the process was successful, the additional data elements for further processing of the message. If the process was not successful, then an errorIndication, possibly with a OID and value pair of an error counter that was incremented.

2.6. Key Localization Algorithm.

A localized key is a secret key shared between a user U and one authoritative SNMP engine E. Even though a user may have only one password and therefore one key for the whole network, the actual secrets shared between the user and each authoritative SNMP engine will be different. This is achieved by key localization [Localized-key].

First, if a user uses a password, then the user's password is converted into a key Ku using one of the two algorithms described in Appendices A.2.1 and A.2.2.

To convert key Ku into a localized key Kul of user U at the authoritative SNMP engine E, one appends the snmpEngineID of the authoritative SNMP engine to the key Ku and then appends the key Ku to the result, thus enveloping the snmpEngineID within the two copies of user's key Ku. Then one runs a secure hash function (which one depends on the authentication protocol defined for this user U at authoritative SNMP engine E; this document defines two authentication protocols with their associated algorithms based on MD5 and SHA). The output of the hash-function is the localized key Kul for user U at the authoritative SNMP engine E.

3. Elements of Procedure

This section describes the security related procedures followed by an SNMP engine when processing SNMP messages according to the User-based Security Model.

3.1. Generating an Outgoing SNMP Message

This section describes the procedure followed by an SNMP engine whenever it generates a message containing a management operation (like a request, a response, a notification, or a report) on behalf of a user, with a particular securityLevel.

1) a) If any securityStateReference is passed (Response or Report message), then information concerning the user is extracted from the cachedSecurityData. The cachedSecurityData can now be discarded. The securityEngineID is set to the local snmpEngineID. The securityLevel is set to the value specified by the calling module.

   Otherwise,

   b) based on the securityName, information concerning the user at the destination snmpEngineID, specified by the securityEngineID, is extracted from the Local Configuration Datastore (LCD, usmUserTable). If information about the user is absent from the LCD, then an error indication (unknownSecurityName) is returned to the calling module.

2) If the securityLevel specifies that the message is to be protected from disclosure, but the user does not support both an authentication and a privacy protocol then the message cannot be sent. An error indication (unsupportedSecurityLevel) is returned to the calling module.

3) If the securityLevel specifies that the message is to be authenticated, but the user does not support an authentication protocol, then the message cannot be sent. An error indication (unsupportedSecurityLevel) is returned to the calling module.

4) a) If the securityLevel specifies that the message is to be protected from disclosure, then the octet sequence representing the serialized scopedPDU is encrypted according to the user's privacy protocol. To do so a call is made to the privacy module that implements the user's privacy protocol according to the abstract primitive:

```
statusInformation =       -- success or failure
  encryptData(
  IN    encryptKey        -- user's localized privKey
  IN    dataToEncrypt     -- serialized scopedPDU
  OUT   encryptedData     -- serialized encryptedPDU
  OUT   privParameters    -- serialized privacy parameters
        )
```

   statusInformation
   indicates if the encryption process was successful or not.

   encryptKey
   the user's localized private privKey is the secret key that can be used by the encryption algorithm.

dataToEncrypt
the serialized scopedPDU is the data to be encrypted.

encryptedData
the encryptedPDU represents the encrypted scopedPDU, encoded as an OCTET STRING.

privParameters
the privacy parameters, encoded as an OCTET STRING.

If the privacy module returns failure, then the message cannot be sent and an error indication (encryptionError) is returned to the calling module.

If the privacy module returns success, then the returned privParameters are put into the msgPrivacyParameters field of the securityParameters and the encryptedPDU serves as the payload of the message being prepared.

Otherwise,

b) If the securityLevel specifies that the message is not to be be protected from disclosure, then a zero-length OCTET STRING is encoded into the msgPrivacyParameters field of the securityParameters and the plaintext scopedPDU serves as the payload of the message being prepared.

5) The securityEngineID is encoded as an OCTET STRING into the msgAuthoritativeEngineID field of the securityParameters. Note that an empty (zero length) securityEngineID is OK for a Request message, because that will cause the remote (authoritative) SNMP engine to return a Report PDU with the proper securityEngineID included in the msgAuthoritativeEngineID in the securityParameters of that returned Report PDU.

6) a) If the securityLevel specifies that the message is to be authenticated, then the current values of snmpEngineBoots and snmpEngineTime corresponding to the securityEngineID from the LCD are used.

Otherwise,

b) If this is a Response or Report message, then the current value of snmpEngineBoots and snmpEngineTime corresponding to the local snmpEngineID from the LCD are used.

Otherwise,

c) If this is a Request message, then a zero value is used for both snmpEngineBoots and snmpEngineTime. This zero value gets used if snmpEngineID is empty.

The values are encoded as INTEGER respectively into the msgAuthoritativeEngineBoots and msgAuthoritativeEngineTime fields of the securityParameters.

7) The userName is encoded as an OCTET STRING into the msgUserName field of the securityParameters.

8) a) If the securityLevel specifies that the message is to be authenticated, the message is authenticated according to the user's authentication protocol. To do so a call is made to the authentication module that implements the user's authentication protocol according to the abstract service primitive:

```
statusInformation =
  authenticateOutgoingMsg(
  IN  authKey               -- the user's localized authKey
  IN  wholeMsg              -- unauthenticated message
  OUT authenticatedWholeMsg -- authenticated complete message
      )
```

statusInformation
indicates if authentication was successful or not.

authKey
the user's localized private authKey is the secret key that can be used by the authentication algorithm.

wholeMsg
the complete serialized message to be authenticated.

authenticatedWholeMsg
the same as the input given to the authenticateOutgoingMsg service, but with msgAuthenticationParameters properly filled in.

If the authentication module returns failure, then the message cannot be sent and an error indication (authenticationFailure) is returned to the calling module.

If the authentication module returns success, then the msgAuthenticationParameters field is put into the securityParameters and the authenticatedWholeMsg represents the serialization of the authenticated message being prepared.

Otherwise,

b) If the securityLevel specifies that the message is not to be authenticated then a zero-length OCTET STRING is encoded into the msgAuthenticationParameters field of the securityParameters. The wholeMsg is now serialized and then represents the unauthenticated message being prepared.

9) The completed message with its length is returned to the calling module with the statusInformation set to success.

3.2. Processing an Incoming SNMP Message

This section describes the procedure followed by an SNMP engine whenever it receives a message containing a management operation on behalf of a user, with a particular securityLevel.

To simplify the elements of procedure, the release of state information is not always explicitly specified. As a general rule, if state information is available when a message gets discarded, the state information should also be released. Also, an error indication can return an OID and value for an incremented counter and optionally a value for securityLevel, and values for contextEngineID or contextName for the counter. In addition, the securityStateReference data is returned if any such information is available at the point where the error is detected.

1) If the received securityParameters is not the serialization (according to the conventions of [RFC3417]) of an OCTET STRING formatted according to the UsmSecurityParameters defined in section 2.4, then the snmpInASNParseErrs counter [RFC3418] is incremented, and an error indication (parseError) is returned to the calling module. Note that we return without the OID and value of the incremented counter, because in this case there is not enough information to generate a Report PDU.

2) The values of the security parameter fields are extracted from the securityParameters. The securityEngineID to be returned to the caller is the value of the msgAuthoritativeEngineID field. The cachedSecurityData is prepared and a securityStateReference is prepared to reference this data. Values to be cached are:

msgUserName

3) If the value of the msgAuthoritativeEngineID field in the securityParameters is unknown then:

   a) a non-authoritative SNMP engine that performs discovery may optionally create a new entry in its Local Configuration Datastore (LCD) and continue processing;

      or

   b) the usmStatsUnknownEngineIDs counter is incremented, and an error indication (unknownEngineID) together with the OID and value of the incremented counter is returned to the calling module.

   Note in the event that a zero-length, or other illegally sized msgAuthoritativeEngineID is received, b) should be chosen to facilitate engineID discovery. Otherwise the choice between a) and b) is an implementation issue.

4) Information about the value of the msgUserName and msgAuthoritativeEngineID fields is extracted from the Local Configuration Datastore (LCD, usmUserTable). If no information is available for the user, then the usmStatsUnknownUserNames counter is incremented and an error indication (unknownSecurityName) together with the OID and value of the incremented counter is returned to the calling module.

5) If the information about the user indicates that it does not support the securityLevel requested by the caller, then the usmStatsUnsupportedSecLevels counter is incremented and an error indication (unsupportedSecurityLevel) together with the OID and value of the incremented counter is returned to the calling module.

6) If the securityLevel specifies that the message is to be authenticated, then the message is authenticated according to the user's authentication protocol. To do so a call is made to the authentication module that implements the user's authentication protocol according to the abstract service primitive:

```
statusInformation =          -- success or failure
  authenticateIncomingMsg(
  IN   authKey               -- the user's localized authKey
  IN   authParameters        -- as received on the wire
  IN   wholeMsg              -- as received on the wire
  OUT  authenticatedWholeMsg -- checked for authentication
       )
```

statusInformation
indicates if authentication was successful or not.

authKey
the user's localized private authKey is the secret key that can be used by the authentication algorithm.

wholeMsg
the complete serialized message to be authenticated.

authenticatedWholeMsg
the same as the input given to the authenticateIncomingMsg service, but after authentication has been checked.

If the authentication module returns failure, then the message cannot be trusted, so the usmStatsWrongDigests counter is incremented and an error indication (authenticationFailure) together with the OID and value of the incremented counter is returned to the calling module.

If the authentication module returns success, then the message is authentic and can be trusted so processing continues.

7) If the securityLevel indicates an authenticated message, then the local values of snmpEngineBoots, snmpEngineTime and latestReceivedEngineTime corresponding to the value of the msgAuthoritativeEngineID field are extracted from the Local Configuration Datastore.

a) If the extracted value of msgAuthoritativeEngineID is the same as the value of snmpEngineID of the processing SNMP engine (meaning this is the authoritative SNMP engine), then if any of the following conditions is true, then the message is considered to be outside of the Time Window:

- the local value of snmpEngineBoots is 2147483647;

- the value of the msgAuthoritativeEngineBoots field differs from the local value of snmpEngineBoots; or,

- the value of the msgAuthoritativeEngineTime field differs from the local notion of snmpEngineTime by more than +/- 150 seconds.

If the message is considered to be outside of the Time Window then the usmStatsNotInTimeWindows counter is incremented and an error indication (notInTimeWindow) together with the OID, the value of the incremented counter, and an indication that

the error must be reported with a securityLevel of authNoPriv, is returned to the calling module

b) If the extracted value of msgAuthoritativeEngineID is not the same as the value snmpEngineID of the processing SNMP engine (meaning this is not the authoritative SNMP engine), then:

1) if at least one of the following conditions is true:

- the extracted value of the msgAuthoritativeEngineBoots field is greater than the local notion of the value of snmpEngineBoots; or,

- the extracted value of the msgAuthoritativeEngineBoots field is equal to the local notion of the value of snmpEngineBoots, and the extracted value of msgAuthoritativeEngineTime field is greater than the value of latestReceivedEngineTime,

then the LCD entry corresponding to the extracted value of the msgAuthoritativeEngineID field is updated, by setting:

- the local notion of the value of snmpEngineBoots to the value of the msgAuthoritativeEngineBoots field,

- the local notion of the value of snmpEngineTime to the value of the msgAuthoritativeEngineTime field, and

- the latestReceivedEngineTime to the value of the value of the msgAuthoritativeEngineTime field.

2) if any of the following conditions is true, then the message is considered to be outside of the Time Window:

- the local notion of the value of snmpEngineBoots is 2147483647;

- the value of the msgAuthoritativeEngineBoots field is less than the local notion of the value of snmpEngineBoots; or,

- the value of the msgAuthoritativeEngineBoots field is equal to the local notion of the value of snmpEngineBoots and the value of the msgAuthoritativeEngineTime field is more than 150 seconds less than the local notion of the value of snmpEngineTime.

If the message is considered to be outside of the Time Window then an error indication (notInTimeWindow) is returned to the calling module.

Note that this means that a too old (possibly replayed) message has been detected and is deemed unauthentic.

Note that this procedure allows for the value of msgAuthoritativeEngineBoots in the message to be greater than the local notion of the value of snmpEngineBoots to allow for received messages to be accepted as authentic when received from an authoritative SNMP engine that has re-booted since the receiving SNMP engine last (re-)synchronized.

8) a) If the securityLevel indicates that the message was protected from disclosure, then the OCTET STRING representing the encryptedPDU is decrypted according to the user's privacy protocol to obtain an unencrypted serialized scopedPDU value. To do so a call is made to the privacy module that implements the user's privacy protocol according to the abstract primitive:

```
statusInformation =       -- success or failure
  decryptData(
  IN    decryptKey        -- the user's localized privKey
  IN    privParameters    -- as received on the wire
  IN    encryptedData     -- encryptedPDU as received
  OUT   decryptedData     -- serialized decrypted scopedPDU
        )
```

statusInformation
indicates if the decryption process was successful or not.

decryptKey
the user's localized private privKey is the secret key that can be used by the decryption algorithm.

privParameters
the msgPrivacyParameters, encoded as an OCTET STRING.

encryptedData
the encryptedPDU represents the encrypted scopedPDU, encoded as an OCTET STRING.

decryptedData
the serialized scopedPDU if decryption is successful.

If the privacy module returns failure, then the message can not be processed, so the usmStatsDecryptionErrors counter is incremented and an error indication (decryptionError) together with the OID and value of the incremented counter is returned to the calling module.

If the privacy module returns success, then the decrypted scopedPDU is the message payload to be returned to the calling module.

Otherwise,

b) The scopedPDU component is assumed to be in plain text and is the message payload to be returned to the calling module.

9) The maxSizeResponseScopedPDU is calculated. This is the maximum size allowed for a scopedPDU for a possible Response message. Provision is made for a message header that allows the same securityLevel as the received Request.

10) The securityName for the user is retrieved from the usmUserTable.

11) The security data is cached as cachedSecurityData, so that a possible response to this message can and will use the same authentication and privacy secrets. Information to be saved/cached is as follows:

msgUserName,
usmUserAuthProtocol, usmUserAuthKey
usmUserPrivProtocol, usmUserPrivKey

12) The statusInformation is set to success and a return is made to the calling module passing back the OUT parameters as specified in the processIncomingMsg primitive.

4. Discovery

The User-based Security Model requires that a discovery process obtains sufficient information about other SNMP engines in order to communicate with them. Discovery requires an non-authoritative SNMP engine to learn the authoritative SNMP engine's snmpEngineID value before communication may proceed. This may be accomplished by generating a Request message with a securityLevel of noAuthNoPriv, a msgUserName of zero-length, a msgAuthoritativeEngineID value of zero length, and the varBindList left empty. The response to this message will be a Report message containing the snmpEngineID of the authoritative SNMP engine as the value of the msgAuthoritativeEngineID field within the msgSecurityParameters

field. It contains a Report PDU with the usmStatsUnknownEngineIDs counter in the varBindList.

If authenticated communication is required, then the discovery process should also establish time synchronization with the authoritative SNMP engine. This may be accomplished by sending an authenticated Request message with the value of msgAuthoritativeEngineID set to the newly learned snmpEngineID and with the values of msgAuthoritativeEngineBoots and msgAuthoritativeEngineTime set to zero. For an authenticated Request message, a valid userName must be used in the msgUserName field. The response to this authenticated message will be a Report message containing the up to date values of the authoritative SNMP engine's snmpEngineBoots and snmpEngineTime as the value of the msgAuthoritativeEngineBoots and msgAuthoritativeEngineTime fields respectively. It also contains the usmStatsNotInTimeWindows counter in the varBindList of the Report PDU. The time synchronization then happens automatically as part of the procedures in section 3.2 step 7b. See also section 2.3.

5. Definitions

```
SNMP-USER-BASED-SM-MIB DEFINITIONS ::= BEGIN

IMPORTS
    MODULE-IDENTITY, OBJECT-TYPE,
    OBJECT-IDENTITY,
    snmpModules, Counter32                FROM SNMPv2-SMI
    TEXTUAL-CONVENTION, TestAndIncr,
    RowStatus, RowPointer,
    StorageType, AutonomousType           FROM SNMPv2-TC
    MODULE-COMPLIANCE, OBJECT-GROUP       FROM SNMPv2-CONF
    SnmpAdminString, SnmpEngineID,
    snmpAuthProtocols, snmpPrivProtocols  FROM SNMP-FRAMEWORK-MIB;

snmpUsmMIB MODULE-IDENTITY
    LAST-UPDATED "200210160000Z"            -- 16 Oct 2002, midnight
    ORGANIZATION "SNMPv3 Working Group"
    CONTACT-INFO "WG-email:   snmpv3@lists.tislabs.com
                  Subscribe:  majordomo@lists.tislabs.com
                              In msg body:  subscribe snmpv3

                  Chair:      Russ Mundy
                              Network Associates Laboratories
                  postal:     15204 Omega Drive, Suite 300
                              Rockville, MD 20850-4601
                              USA
                  email:      mundy@tislabs.com
```

```
                    phone:      +1 301-947-7107

                    Co-Chair:   David Harrington
                                Enterasys Networks
                    Postal:     35 Industrial Way
                                P. O. Box 5004
                                Rochester, New Hampshire 03866-5005
                                USA
                    EMail:      dbh@enterasys.com
                    Phone:      +1 603-337-2614

                    Co-editor   Uri Blumenthal
                                Lucent Technologies
                    postal:     67 Whippany Rd.
                                Whippany, NJ 07981
                                USA
                    email:      uri@lucent.com
                    phone:      +1-973-386-2163

                    Co-editor:  Bert Wijnen
                                Lucent Technologies
                    postal:     Schagen 33
                                3461 GL Linschoten
                                Netherlands
                    email:      bwijnen@lucent.com
                    phone:      +31-348-480-685
                   "
    DESCRIPTION  "The management information definitions for the
                  SNMP User-based Security Model.

                  Copyright (C) The Internet Society (2002). This
                  version of this MIB module is part of RFC 3414;
                  see the RFC itself for full legal notices.
                 "
--  Revision history

    REVISION     "200210160000Z"          -- 16 Oct 2002, midnight
    DESCRIPTION  "Changes in this revision:
                  - Updated references and contact info.
                  - Clarification to usmUserCloneFrom DESCRIPTION
                    clause
                  - Fixed 'command responder' into 'command generator'
                    in last para of DESCRIPTION clause of
                    usmUserTable.
                  This revision published as RFC3414.
                 "
    REVISION     "199901200000Z"          -- 20 Jan 1999, midnight
    DESCRIPTION  "Clarifications, published as RFC2574"
```

```
    REVISION     "199711200000Z"          -- 20 Nov 1997, midnight
    DESCRIPTION  "Initial version, published as RFC2274"

    ::= { snmpModules 15 }

-- Administrative assignments ****************************************

usmMIBObjects     OBJECT IDENTIFIER ::= { snmpUsmMIB 1 }
usmMIBConformance OBJECT IDENTIFIER ::= { snmpUsmMIB 2 }

-- Identification of Authentication and Privacy Protocols ************

usmNoAuthProtocol OBJECT-IDENTITY
    STATUS        current
    DESCRIPTION  "No Authentication Protocol."
    ::= { snmpAuthProtocols 1 }

usmHMACMD5AuthProtocol OBJECT-IDENTITY
    STATUS        current
    DESCRIPTION  "The HMAC-MD5-96 Digest Authentication Protocol."
    REFERENCE    "- H. Krawczyk, M. Bellare, R. Canetti HMAC:
                    Keyed-Hashing for Message Authentication,
                    RFC2104, Feb 1997.
                  - Rivest, R., Message Digest Algorithm MD5, RFC1321.
                 "
    ::= { snmpAuthProtocols 2 }

usmHMACSHAAuthProtocol OBJECT-IDENTITY
    STATUS        current
    DESCRIPTION  "The HMAC-SHA-96 Digest Authentication Protocol."
    REFERENCE    "- H. Krawczyk, M. Bellare, R. Canetti, HMAC:
                    Keyed-Hashing for Message Authentication,
                    RFC2104, Feb 1997.
                  - Secure Hash Algorithm. NIST FIPS 180-1.
                 "
    ::= { snmpAuthProtocols 3 }

usmNoPrivProtocol OBJECT-IDENTITY
    STATUS        current
    DESCRIPTION  "No Privacy Protocol."
    ::= { snmpPrivProtocols 1 }

usmDESPrivProtocol OBJECT-IDENTITY
    STATUS        current
    DESCRIPTION  "The CBC-DES Symmetric Encryption Protocol."
    REFERENCE    "- Data Encryption Standard, National Institute of
                    Standards and Technology.  Federal Information
                    Processing Standard (FIPS) Publication 46-1.
```

```
                    Supersedes FIPS Publication 46,
                    (January, 1977; reaffirmed January, 1988).

                  - Data Encryption Algorithm, American National
                    Standards Institute.  ANSI X3.92-1981,
                    (December, 1980).

                  - DES Modes of Operation, National Institute of
                    Standards and Technology.  Federal Information
                    Processing Standard (FIPS) Publication 81,
                    (December, 1980).

                  - Data Encryption Algorithm - Modes of Operation,
                    American National Standards Institute.
                    ANSI X3.106-1983, (May 1983).
                "
    ::= { snmpPrivProtocols 2 }

-- Textual Conventions ***********************************************

KeyChange ::=     TEXTUAL-CONVENTION
   STATUS         current
   DESCRIPTION
         "Every definition of an object with this syntax must identify
          a protocol P, a secret key K, and a hash algorithm H
          that produces output of L octets.

          The object's value is a manager-generated, partially-random
          value which, when modified, causes the value of the secret
          key K, to be modified via a one-way function.

          The value of an instance of this object is the concatenation
          of two components: first a 'random' component and then a
          'delta' component.

          The lengths of the random and delta components
          are given by the corresponding value of the protocol P;
          if P requires K to be a fixed length, the length of both the
          random and delta components is that fixed length; if P
          allows the length of K to be variable up to a particular
          maximum length, the length of the random component is that
          maximum length and the length of the delta component is any
          length less than or equal to that maximum length.
          For example, usmHMACMD5AuthProtocol requires K to be a fixed
          length of 16 octets and L - of 16 octets.
          usmHMACSHAAuthProtocol requires K to be a fixed length of
          20 octets and L - of 20 octets. Other protocols may define
          other sizes, as deemed appropriate.
```

When a requester wants to change the old key K to a new key keyNew on a remote entity, the 'random' component is obtained from either a true random generator, or from a pseudorandom generator, and the 'delta' component is computed as follows:

- a temporary variable is initialized to the existing value of K;
- if the length of the keyNew is greater than L octets, then:
  - the random component is appended to the value of the temporary variable, and the result is input to the the hash algorithm H to produce a digest value, and the temporary variable is set to this digest value;
  - the value of the temporary variable is XOR-ed with the first (next) L-octets (16 octets in case of MD5) of the keyNew to produce the first (next) L-octets (16 octets in case of MD5) of the 'delta' component.
  - the above two steps are repeated until the unused portion of the keyNew component is L octets or less,
- the random component is appended to the value of the temporary variable, and the result is input to the hash algorithm H to produce a digest value;
- this digest value, truncated if necessary to be the same length as the unused portion of the keyNew, is XOR-ed with the unused portion of the keyNew to produce the (final portion of the) 'delta' component.

For example, using MD5 as the hash algorithm H:

```
iterations = (lenOfDelta - 1)/16; /* integer division */
temp = keyOld;
for (i = 0; i < iterations; i++) {
    temp = MD5 (temp || random);
    delta[i*16 .. (i*16)+15] =
           temp XOR keyNew[i*16 .. (i*16)+15];
}
temp = MD5 (temp || random);
delta[i*16 .. lenOfDelta-1] =
       temp XOR keyNew[i*16 .. lenOfDelta-1];
```

The 'random' and 'delta' components are then concatenated as described above, and the resulting octet string is sent to the recipient as the new value of an instance of this object.

At the receiver side, when an instance of this object is set to a new value, then a new value of K is computed as follows:

```
          - a temporary variable is initialized to the existing value
            of K;
          - if the length of the delta component is greater than L
            octets, then:
             - the random component is appended to the value of the
               temporary variable, and the result is input to the
               hash algorithm H to produce a digest value, and the
               temporary variable is set to this digest value;
             - the value of the temporary variable is XOR-ed with
               the first (next) L-octets (16 octets in case of MD5)
               of the delta component to produce the first (next)
               L-octets (16 octets in case of MD5) of the new value
               of K.
             - the above two steps are repeated until the unused
               portion of the delta component is L octets or less,
          - the random component is appended to the value of the
            temporary variable, and the result is input to the
            hash algorithm H to produce a digest value;
          - this digest value, truncated if necessary to be the same
            length as the unused portion of the delta component, is
            XOR-ed with the unused portion of the delta component to
            produce the (final portion of the) new value of K.

          For example, using MD5 as the hash algorithm H:

             iterations = (lenOfDelta - 1)/16; /* integer division */
             temp = keyOld;
             for (i = 0; i < iterations; i++) {
                 temp = MD5 (temp || random);
                 keyNew[i*16 .. (i*16)+15] =
                        temp XOR delta[i*16 .. (i*16)+15];
             }
             temp = MD5 (temp || random);
             keyNew[i*16 .. lenOfDelta-1] =
                    temp XOR delta[i*16 .. lenOfDelta-1];

          The value of an object with this syntax, whenever it is
          retrieved by the management protocol, is always the zero
          length string.

          Note that the keyOld and keyNew are the localized keys.

          Note that it is probably wise that when an SNMP entity sends
          a SetRequest to change a key, that it keeps a copy of the old
          key until it has confirmed that the key change actually
          succeeded.
        "
    SYNTAX       OCTET STRING
```

```
-- Statistics for the User-based Security Model **********************

usmStats         OBJECT IDENTIFIER ::= { usmMIBObjects 1 }


usmStatsUnsupportedSecLevels OBJECT-TYPE
    SYNTAX       Counter32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION "The total number of packets received by the SNMP
                 engine which were dropped because they requested a
                 securityLevel that was unknown to the SNMP engine
                 or otherwise unavailable.
                "
    ::= { usmStats 1 }

usmStatsNotInTimeWindows OBJECT-TYPE
    SYNTAX       Counter32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION "The total number of packets received by the SNMP
                 engine which were dropped because they appeared
                 outside of the authoritative SNMP engine's window.
                "
    ::= { usmStats 2 }

usmStatsUnknownUserNames OBJECT-TYPE
    SYNTAX       Counter32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION "The total number of packets received by the SNMP
                 engine which were dropped because they referenced a
                 user that was not known to the SNMP engine.
                "
    ::= { usmStats 3 }

usmStatsUnknownEngineIDs OBJECT-TYPE
    SYNTAX       Counter32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION "The total number of packets received by the SNMP
                 engine which were dropped because they referenced an
                 snmpEngineID that was not known to the SNMP engine.
                "
    ::= { usmStats 4 }

usmStatsWrongDigests OBJECT-TYPE
```

```
    SYNTAX       Counter32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION "The total number of packets received by the SNMP
                 engine which were dropped because they didn't
                 contain the expected digest value.
                "
    ::= { usmStats 5 }

usmStatsDecryptionErrors OBJECT-TYPE
    SYNTAX       Counter32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION "The total number of packets received by the SNMP
                 engine which were dropped because they could not be
                 decrypted.
                "
    ::= { usmStats 6 }

-- The usmUser Group ***********************************************

usmUser          OBJECT IDENTIFIER ::= { usmMIBObjects 2 }

usmUserSpinLock  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 facilities to alter secrets in the
                 usmUserTable.
                "
    ::= { usmUser 1 }

-- The table of valid users for the User-based Security Model ********

usmUserTable     OBJECT-TYPE
    SYNTAX       SEQUENCE OF UsmUserEntry
    MAX-ACCESS   not-accessible
    STATUS       current
    DESCRIPTION "The table of users configured in the SNMP engine's
                 Local Configuration Datastore (LCD).

                 To create a new user (i.e., to instantiate a new
                 conceptual row in this table), it is recommended to
                 follow this procedure:

                   1)  GET(usmUserSpinLock.0) and save in sValue.
```

```
  2)  SET(usmUserSpinLock.0=sValue,
          usmUserCloneFrom=templateUser,
          usmUserStatus=createAndWait)
      You should use a template user to clone from
      which has the proper auth/priv protocol defined.

If the new user is to use privacy:

  3)  generate the keyChange value based on the secret
      privKey of the clone-from user and the secret key
      to be used for the new user. Let us call this
      pkcValue.
  4)  GET(usmUserSpinLock.0) and save in sValue.
  5)  SET(usmUserSpinLock.0=sValue,
          usmUserPrivKeyChange=pkcValue
          usmUserPublic=randomValue1)
  6)  GET(usmUserPulic) and check it has randomValue1.
      If not, repeat steps 4-6.

If the new user will never use privacy:

  7)  SET(usmUserPrivProtocol=usmNoPrivProtocol)

If the new user is to use authentication:

  8)  generate the keyChange value based on the secret
      authKey of the clone-from user and the secret key
      to be used for the new user. Let us call this
      akcValue.
  9)  GET(usmUserSpinLock.0) and save in sValue.
  10) SET(usmUserSpinLock.0=sValue,
          usmUserAuthKeyChange=akcValue
          usmUserPublic=randomValue2)
  11) GET(usmUserPulic) and check it has randomValue2.
      If not, repeat steps 9-11.

If the new user will never use authentication:

  12) SET(usmUserAuthProtocol=usmNoAuthProtocol)

Finally, activate the new user:

  13) SET(usmUserStatus=active)

The new user should now be available and ready to be
used for SNMPv3 communication. Note however that access
to MIB data must be provided via configuration of the
SNMP-VIEW-BASED-ACM-MIB.
```

```
                The use of usmUserSpinlock is to avoid conflicts with
                another SNMP command generator application which may
                also be acting on the usmUserTable.
               "
    ::= { usmUser 2 }

usmUserEntry     OBJECT-TYPE
    SYNTAX       UsmUserEntry
    MAX-ACCESS   not-accessible
    STATUS       current
    DESCRIPTION "A user configured in the SNMP engine's Local
                 Configuration Datastore (LCD) for the User-based
                 Security Model.
                "
    INDEX       { usmUserEngineID,
                  usmUserName
                }
    ::= { usmUserTable 1 }

UsmUserEntry ::= SEQUENCE
    {
        usmUserEngineID         SnmpEngineID,
        usmUserName             SnmpAdminString,
        usmUserSecurityName     SnmpAdminString,
        usmUserCloneFrom        RowPointer,
        usmUserAuthProtocol     AutonomousType,
        usmUserAuthKeyChange    KeyChange,
        usmUserOwnAuthKeyChange KeyChange,
        usmUserPrivProtocol     AutonomousType,
        usmUserPrivKeyChange    KeyChange,
        usmUserOwnPrivKeyChange KeyChange,
        usmUserPublic           OCTET STRING,
        usmUserStorageType      StorageType,
        usmUserStatus           RowStatus
    }

usmUserEngineID  OBJECT-TYPE
    SYNTAX       SnmpEngineID
    MAX-ACCESS   not-accessible
    STATUS       current
    DESCRIPTION "An SNMP engine's administratively-unique identifier.

                 In a simple agent, this value is always that agent's
                 own snmpEngineID value.

                 The value can also take the value of the snmpEngineID
                 of a remote SNMP engine with which this user can
                 communicate.
```

```
                "
    ::= { usmUserEntry 1 }

usmUserName      OBJECT-TYPE
    SYNTAX       SnmpAdminString (SIZE(1..32))
    MAX-ACCESS   not-accessible
    STATUS       current
    DESCRIPTION "A human readable string representing the name of
                 the user.

                 This is the (User-based Security) Model dependent
                 security ID.
                "
    ::= { usmUserEntry 2 }

usmUserSecurityName OBJECT-TYPE
    SYNTAX       SnmpAdminString
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION "A human readable string representing the user in
                 Security Model independent format.

                 The default transformation of the User-based Security
                 Model dependent security ID to the securityName and
                 vice versa is the identity function so that the
                 securityName is the same as the userName.
                "
    ::= { usmUserEntry 3 }

usmUserCloneFrom OBJECT-TYPE
    SYNTAX       RowPointer
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION "A pointer to another conceptual row in this
                 usmUserTable.  The user in this other conceptual
                 row is called the clone-from user.

                 When a new user is created (i.e., a new conceptual
                 row is instantiated in this table), the privacy and
                 authentication parameters of the new user must be
                 cloned from its clone-from user. These parameters are:
                   - authentication protocol (usmUserAuthProtocol)
                   - privacy protocol (usmUserPrivProtocol)
                 They will be copied regardless of what the current
                 value is.

                 Cloning also causes the initial values of the secret
                 authentication key (authKey) and the secret encryption
```

```
                key (privKey) of the new user to be set to the same
                values as the corresponding secrets of the clone-from
                user to allow the KeyChange process to occur as
                required during user creation.

                The first time an instance of this object is set by
                a management operation (either at or after its
                instantiation), the cloning process is invoked.
                Subsequent writes are successful but invoke no
                action to be taken by the receiver.
                The cloning process fails with an 'inconsistentName'
                error if the conceptual row representing the
                clone-from user does not exist or is not in an active
                state when the cloning process is invoked.

                When this object is read, the ZeroDotZero OID
                is returned.
               "
    ::= { usmUserEntry 4 }

usmUserAuthProtocol OBJECT-TYPE
    SYNTAX       AutonomousType
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION "An indication of whether messages sent on behalf of
                this user to/from the SNMP engine identified by
                usmUserEngineID, can be authenticated, and if so,
                the type of authentication protocol which is used.

                An instance of this object is created concurrently
                with the creation of any other object instance for
                the same user (i.e., as part of the processing of
                the set operation which creates the first object
                instance in the same conceptual row).

                If an initial set operation (i.e. at row creation time)
                tries to set a value for an unknown or unsupported
                protocol, then a 'wrongValue' error must be returned.

                The value will be overwritten/set when a set operation
                is performed on the corresponding instance of
                usmUserCloneFrom.

                Once instantiated, the value of such an instance of
                this object can only be changed via a set operation to
                the value of the usmNoAuthProtocol.

                If a set operation tries to change the value of an
```

```
                existing instance of this object to any value other
                than usmNoAuthProtocol, then an 'inconsistentValue'
                error must be returned.

                If a set operation tries to set the value to the
                usmNoAuthProtocol while the usmUserPrivProtocol value
                in the same row is not equal to usmNoPrivProtocol,
                then an 'inconsistentValue' error must be returned.
                That means that an SNMP command generator application
                must first ensure that the usmUserPrivProtocol is set
                to the usmNoPrivProtocol value before it can set
                the usmUserAuthProtocol value to usmNoAuthProtocol.
               "
    DEFVAL      { usmNoAuthProtocol }
    ::= { usmUserEntry 5 }

usmUserAuthKeyChange OBJECT-TYPE
    SYNTAX       KeyChange   -- typically (SIZE (0 | 32)) for HMACMD5
                             -- typically (SIZE (0 | 40)) for HMACSHA
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION "An object, which when modified, causes the secret
                 authentication key used for messages sent on behalf
                 of this user to/from the SNMP engine identified by
                 usmUserEngineID, to be modified via a one-way
                 function.

                 The associated protocol is the usmUserAuthProtocol.
                 The associated secret key is the user's secret
                 authentication key (authKey). The associated hash
                 algorithm is the algorithm used by the user's
                 usmUserAuthProtocol.

                 When creating a new user, it is an 'inconsistentName'
                 error for a set operation to refer to this object
                 unless it is previously or concurrently initialized
                 through a set operation on the corresponding instance
                 of usmUserCloneFrom.

                 When the value of the corresponding usmUserAuthProtocol
                 is usmNoAuthProtocol, then a set is successful, but
                 effectively is a no-op.

                 When this object is read, the zero-length (empty)
                 string is returned.

                 The recommended way to do a key change is as follows:
```

```
                  1) GET(usmUserSpinLock.0) and save in sValue.
                  2) generate the keyChange value based on the old
                     (existing) secret key and the new secret key,
                     let us call this kcValue.

                If you do the key change on behalf of another user:

                  3) SET(usmUserSpinLock.0=sValue,
                         usmUserAuthKeyChange=kcValue
                         usmUserPublic=randomValue)

                If you do the key change for yourself:

                  4) SET(usmUserSpinLock.0=sValue,
                         usmUserOwnAuthKeyChange=kcValue
                         usmUserPublic=randomValue)

                If you get a response with error-status of noError,
                then the SET succeeded and the new key is active.
                If you do not get a response, then you can issue a
                GET(usmUserPublic) and check if the value is equal
                to the randomValue you did send in the SET. If so, then
                the key change succeeded and the new key is active
                (probably the response got lost). If not, then the SET
                request probably never reached the target and so you
                can start over with the procedure above.
               "
    DEFVAL      { ''H }    -- the empty string
    ::= { usmUserEntry 6 }

usmUserOwnAuthKeyChange OBJECT-TYPE
    SYNTAX       KeyChange   -- typically (SIZE (0 | 32)) for HMACMD5
                             -- typically (SIZE (0 | 40)) for HMACSHA
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION "Behaves exactly as usmUserAuthKeyChange, with one
                 notable difference: in order for the set operation
                 to succeed, the usmUserName of the operation
                 requester must match the usmUserName that
                 indexes the row which is targeted by this
                 operation.
                 In addition, the USM security model must be
                 used for this operation.

                 The idea here is that access to this column can be
                 public, since it will only allow a user to change
                 his own secret authentication key (authKey).
                 Note that this can only be done once the row is active.
```

```
                When a set is received and the usmUserName of the
                requester is not the same as the umsUserName that
                indexes the row which is targeted by this operation,
                then a 'noAccess' error must be returned.

                When a set is received and the security model in use
                is not USM, then a 'noAccess' error must be returned.
               "
    DEFVAL      { ''H }    -- the empty string
    ::= { usmUserEntry 7 }

usmUserPrivProtocol OBJECT-TYPE
    SYNTAX       AutonomousType
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION "An indication of whether messages sent on behalf of
                this user to/from the SNMP engine identified by
                usmUserEngineID, can be protected from disclosure,
                and if so, the type of privacy protocol which is used.

                An instance of this object is created concurrently
                with the creation of any other object instance for
                the same user (i.e., as part of the processing of
                the set operation which creates the first object
                instance in the same conceptual row).

                If an initial set operation (i.e. at row creation time)
                tries to set a value for an unknown or unsupported
                protocol, then a 'wrongValue' error must be returned.

                The value will be overwritten/set when a set operation
                is performed on the corresponding instance of
                usmUserCloneFrom.

                Once instantiated, the value of such an instance of
                this object can only be changed via a set operation to
                the value of the usmNoPrivProtocol.

                If a set operation tries to change the value of an
                existing instance of this object to any value other
                than usmNoPrivProtocol, then an 'inconsistentValue'
                error must be returned.

                Note that if any privacy protocol is used, then you
                must also use an authentication protocol. In other
                words, if usmUserPrivProtocol is set to anything else
                than usmNoPrivProtocol, then the corresponding instance
                of usmUserAuthProtocol cannot have a value of
```

```
                usmNoAuthProtocol. If it does, then an
                'inconsistentValue' error must be returned.
               "
    DEFVAL      { usmNoPrivProtocol }
    ::= { usmUserEntry 8 }

usmUserPrivKeyChange OBJECT-TYPE
    SYNTAX       KeyChange  -- typically (SIZE (0 | 32)) for DES
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION "An object, which when modified, causes the secret
                 encryption key used for messages sent on behalf
                 of this user to/from the SNMP engine identified by
                 usmUserEngineID, to be modified via a one-way
                 function.

                 The associated protocol is the usmUserPrivProtocol.
                 The associated secret key is the user's secret
                 privacy key (privKey). The associated hash
                 algorithm is the algorithm used by the user's
                 usmUserAuthProtocol.

                 When creating a new user, it is an 'inconsistentName'
                 error for a set operation to refer to this object
                 unless it is previously or concurrently initialized
                 through a set operation on the corresponding instance
                 of usmUserCloneFrom.

                 When the value of the corresponding usmUserPrivProtocol
                 is usmNoPrivProtocol, then a set is successful, but
                 effectively is a no-op.

                 When this object is read, the zero-length (empty)
                 string is returned.
                 See the description clause of usmUserAuthKeyChange for
                 a recommended procedure to do a key change.
                "
    DEFVAL      { ''H }    -- the empty string
    ::= { usmUserEntry 9 }

usmUserOwnPrivKeyChange OBJECT-TYPE
    SYNTAX       KeyChange  -- typically (SIZE (0 | 32)) for DES
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION "Behaves exactly as usmUserPrivKeyChange, with one
                 notable difference: in order for the Set operation
                 to succeed, the usmUserName of the operation
                 requester must match the usmUserName that indexes
```

```
                the row which is targeted by this operation.
                In addition, the USM security model must be
                used for this operation.

                The idea here is that access to this column can be
                public, since it will only allow a user to change
                his own secret privacy key (privKey).
                Note that this can only be done once the row is active.

                When a set is received and the usmUserName of the
                requester is not the same as the umsUserName that
                indexes the row which is targeted by this operation,
                then a 'noAccess' error must be returned.

                When a set is received and the security model in use
                is not USM, then a 'noAccess' error must be returned.
               "
    DEFVAL      { ''H }    -- the empty string
    ::= { usmUserEntry 10 }

usmUserPublic    OBJECT-TYPE
    SYNTAX       OCTET STRING (SIZE(0..32))
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION "A publicly-readable value which can be written as part
                 of the procedure for changing a user's secret
                 authentication and/or privacy key, and later read to
                 determine whether the change of the secret was
                 effected.
                "
    DEFVAL      { ''H }  -- the empty string
    ::= { usmUserEntry 11 }

usmUserStorageType OBJECT-TYPE
    SYNTAX       StorageType
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION "The storage type for this conceptual row.

                 Conceptual rows having the value 'permanent' must
                 allow write-access at a minimum to:

                 - usmUserAuthKeyChange, usmUserOwnAuthKeyChange
                   and usmUserPublic for a user who employs
                   authentication, and
                 - usmUserPrivKeyChange, usmUserOwnPrivKeyChange
                   and usmUserPublic for a user who employs
                   privacy.
```

```
                Note that any user who employs authentication or
                privacy must allow its secret(s) to be updated and
                thus cannot be 'readOnly'.

                If an initial set operation tries to set the value to
                'readOnly' for a user who employs authentication or
                privacy, then an 'inconsistentValue' error must be
                returned.  Note that if the value has been previously
                set (implicit or explicit) to any value, then the rules
                as defined in the StorageType Textual Convention apply.

                It is an implementation issue to decide if a SET for
                a readOnly or permanent row is accepted at all. In some
                contexts this may make sense, in others it may not. If
                a SET for a readOnly or permanent row is not accepted
                at all, then a 'wrongValue' error must be returned.
               "
    DEFVAL      { nonVolatile }
    ::= { usmUserEntry 12 }

usmUserStatus    OBJECT-TYPE
    SYNTAX       RowStatus
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION "The status of this conceptual row.

                Until instances of all corresponding columns are
                appropriately configured, the value of the
                corresponding instance of the usmUserStatus column
                is 'notReady'.

                In particular, a newly created row for a user who
                employs authentication, cannot be made active until the
                corresponding usmUserCloneFrom and usmUserAuthKeyChange
                have been set.

                Further, a newly created row for a user who also
                employs privacy, cannot be made active until the
                usmUserPrivKeyChange has been set.

                The RowStatus TC [RFC2579] requires that this
                DESCRIPTION clause states under which circumstances
                other objects in this row can be modified:

                The value of this object has no effect on whether
                other objects in this conceptual row can be modified,
                except for usmUserOwnAuthKeyChange and
                usmUserOwnPrivKeyChange. For these 2 objects, the
```

```
                value of usmUserStatus MUST be active.
               "
    ::= { usmUserEntry 13 }

-- Conformance Information ****************************************

usmMIBCompliances OBJECT IDENTIFIER ::= { usmMIBConformance 1 }
usmMIBGroups      OBJECT IDENTIFIER ::= { usmMIBConformance 2 }

-- Compliance statements

usmMIBCompliance MODULE-COMPLIANCE
    STATUS       current
    DESCRIPTION "The compliance statement for SNMP engines which
                 implement the SNMP-USER-BASED-SM-MIB.
                "

    MODULE       -- this module
        MANDATORY-GROUPS { usmMIBBasicGroup }

        OBJECT           usmUserAuthProtocol
        MIN-ACCESS       read-only
        DESCRIPTION     "Write access is not required."

        OBJECT           usmUserPrivProtocol
        MIN-ACCESS       read-only
        DESCRIPTION     "Write access is not required."

    ::= { usmMIBCompliances 1 }

-- Units of compliance
usmMIBBasicGroup OBJECT-GROUP
    OBJECTS     {
                  usmStatsUnsupportedSecLevels,
                  usmStatsNotInTimeWindows,
                  usmStatsUnknownUserNames,
                  usmStatsUnknownEngineIDs,
                  usmStatsWrongDigests,
                  usmStatsDecryptionErrors,
                  usmUserSpinLock,
                  usmUserSecurityName,
                  usmUserCloneFrom,
                  usmUserAuthProtocol,
                  usmUserAuthKeyChange,
                  usmUserOwnAuthKeyChange,
                  usmUserPrivProtocol,
                  usmUserPrivKeyChange,
                  usmUserOwnPrivKeyChange,
```

```
                  usmUserPublic,
                  usmUserStorageType,
                  usmUserStatus
                }
    STATUS       current
    DESCRIPTION "A collection of objects providing for configuration
                 of an SNMP engine which implements the SNMP
                 User-based Security Model.
                "
    ::= { usmMIBGroups 1 }

END
```

6. HMAC-MD5-96 Authentication Protocol

This section describes the HMAC-MD5-96 authentication protocol. This authentication protocol is the first defined for the User-based Security Model. It uses MD5 hash-function which is described in [RFC1321], in HMAC mode described in [RFC2104], truncating the output to 96 bits.

This protocol is identified by usmHMACMD5AuthProtocol.

Over time, other authentication protocols may be defined either as a replacement of this protocol or in addition to this protocol.

6.1. Mechanisms

- In support of data integrity, a message digest algorithm is required. A digest is calculated over an appropriate portion of an SNMP message and included as part of the message sent to the recipient.

- In support of data origin authentication and data integrity, a secret value is prepended to SNMP message prior to computing the digest; the calculated digest is partially inserted into the SNMP message prior to transmission, and the prepended value is not transmitted. The secret value is shared by all SNMP engines authorized to originate messages on behalf of the appropriate user.

6.1.1. Digest Authentication Mechanism

The Digest Authentication Mechanism defined in this memo provides for:

- verification of the integrity of a received message, i.e., the message received is the message sent.

The integrity of the message is protected by computing a digest over an appropriate portion of the message. The digest is computed by the originator of the message, transmitted with the message, and verified by the recipient of the message.

- verification of the user on whose behalf the message was generated.

  A secret value known only to SNMP engines authorized to generate messages on behalf of a user is used in HMAC mode (see [RFC2104]). It also recommends the hash-function output used as Message Authentication Code, to be truncated.

This protocol uses the MD5 [RFC1321] message digest algorithm. A 128-bit MD5 digest is calculated in a special (HMAC) way over the designated portion of an SNMP message and the first 96 bits of this digest is included as part of the message sent to the recipient. The size of the digest carried in a message is 12 octets. The size of the private authentication key (the secret) is 16 octets. For the details see section 6.3.

6.2. Elements of the Digest Authentication Protocol

This section contains definitions required to realize the authentication module defined in this section of this memo.

6.2.1. Users

Authentication using this authentication protocol makes use of a defined set of userNames. For any user on whose behalf a message must be authenticated at a particular SNMP engine, that SNMP engine must have knowledge of that user. An SNMP engine that wishes to communicate with another SNMP engine must also have knowledge of a user known to that engine, including knowledge of the applicable attributes of that user.

A user and its attributes are defined as follows:

<userName>
A string representing the name of the user.
<authKey>
A user’s secret key to be used when calculating a digest.
It MUST be 16 octets long for MD5.

6.2.2. msgAuthoritativeEngineID

The msgAuthoritativeEngineID value contained in an authenticated message specifies the authoritative SNMP engine for that particular message (see the definition of SnmpEngineID in the SNMP Architecture document [RFC3411]).

The user's (private) authentication key is normally different at each authoritative SNMP engine and so the snmpEngineID is used to select the proper key for the authentication process.

6.2.3. SNMP Messages Using this Authentication Protocol

Messages using this authentication protocol carry a msgAuthenticationParameters field as part of the msgSecurityParameters. For this protocol, the msgAuthenticationParameters field is the serialized OCTET STRING representing the first 12 octets of the HMAC-MD5-96 output done over the wholeMsg.

The digest is calculated over the wholeMsg so if a message is authenticated, that also means that all the fields in the message are intact and have not been tampered with.

6.2.4. Services provided by the HMAC-MD5-96 Authentication Module

This section describes the inputs and outputs that the HMAC-MD5-96 Authentication module expects and produces when the User-based Security module calls the HMAC-MD5-96 Authentication module for services.

6.2.4.1. Services for Generating an Outgoing SNMP Message

The HMAC-MD5-96 authentication protocol assumes that the selection of the authKey is done by the caller and that the caller passes the secret key to be used.

Upon completion the authentication module returns statusInformation and, if the message digest was correctly calculated, the wholeMsg with the digest inserted at the proper place. The abstract service primitive is:

```
statusInformation =              -- success or failure
  authenticateOutgoingMsg(
  IN   authKey                   -- secret key for authentication
  IN   wholeMsg                  -- unauthenticated complete message
  OUT  authenticatedWholeMsg     -- complete authenticated message
       )
```

The abstract data elements are:

statusInformation
An indication of whether the authentication process was successful. If not it is an indication of the problem.

authKey
The secret key to be used by the authentication algorithm. The length of this key MUST be 16 octets.

wholeMsg
The message to be authenticated.

authenticatedWholeMsg
The authenticated message (including inserted digest) on output.

Note, that authParameters field is filled by the authentication module and this module and this field should be already present in the wholeMsg before the Message Authentication Code (MAC) is generated.

6.2.4.2. Services for Processing an Incoming SNMP Message

The HMAC-MD5-96 authentication protocol assumes that the selection of the authKey is done by the caller and that the caller passes the secret key to be used.

Upon completion the authentication module returns statusInformation and, if the message digest was correctly calculated, the wholeMsg as it was processed. The abstract service primitive is:

```
statusInformation =              -- success or failure
  authenticateIncomingMsg(
  IN   authKey                   -- secret key for authentication
  IN   authParameters            -- as received on the wire
  IN   wholeMsg                  -- as received on the wire
  OUT  authenticatedWholeMsg     -- complete authenticated message
       )
```

The abstract data elements are:

statusInformation
An indication of whether the authentication process was successful. If not it is an indication of the problem.

authKey
The secret key to be used by the authentication algorithm. The length of this key MUST be 16 octets.

authParameters
The authParameters from the incoming message.

wholeMsg
The message to be authenticated on input and the authenticated message on output.

authenticatedWholeMsg
The whole message after the authentication check is complete.

6.3. Elements of Procedure

This section describes the procedures for the HMAC-MD5-96 authentication protocol.

6.3.1. Processing an Outgoing Message

This section describes the procedure followed by an SNMP engine whenever it must authenticate an outgoing message using the usmHMACMD5AuthProtocol.

1) The msgAuthenticationParameters field is set to the serialization, according to the rules in [RFC3417], of an OCTET STRING containing 12 zero octets.

2) From the secret authKey, two keys K1 and K2 are derived:

   a) extend the authKey to 64 octets by appending 48 zero octets; save it as extendedAuthKey

   b) obtain IPAD by replicating the octet 0x36 64 times;

   c) obtain K1 by XORing extendedAuthKey with IPAD;

   d) obtain OPAD by replicating the octet 0x5C 64 times;

   e) obtain K2 by XORing extendedAuthKey with OPAD.

3) Prepend K1 to the wholeMsg and calculate MD5 digest over it according to [RFC1321].

4) Prepend K2 to the result of the step 4 and calculate MD5 digest over it according to [RFC1321]. Take the first 12 octets of the final digest - this is Message Authentication Code (MAC).

5) Replace the msgAuthenticationParameters field with MAC obtained in the step 4.

6) The authenticatedWholeMsg is then returned to the caller together with statusInformation indicating success.

6.3.2. Processing an Incoming Message

This section describes the procedure followed by an SNMP engine whenever it must authenticate an incoming message using the usmHMACMD5AuthProtocol.

1) If the digest received in the msgAuthenticationParameters field is not 12 octets long, then an failure and an errorIndication (authenticationError) is returned to the calling module.

2) The MAC received in the msgAuthenticationParameters field is saved.

3) The digest in the msgAuthenticationParameters field is replaced by the 12 zero octets.

4) From the secret authKey, two keys K1 and K2 are derived:

   a) extend the authKey to 64 octets by appending 48 zero octets; save it as extendedAuthKey

   b) obtain IPAD by replicating the octet 0x36 64 times;

   c) obtain K1 by XORing extendedAuthKey with IPAD;

   d) obtain OPAD by replicating the octet 0x5C 64 times;

   e) obtain K2 by XORing extendedAuthKey with OPAD.

5) The MAC is calculated over the wholeMsg:

   a) prepend K1 to the wholeMsg and calculate the MD5 digest over it;

   b) prepend K2 to the result of step 5.a and calculate the MD5 digest over it;

   c) first 12 octets of the result of step 5.b is the MAC.

   The msgAuthenticationParameters field is replaced with the MAC value that was saved in step 2.

6) Then the newly calculated MAC is compared with the MAC saved in step 2. If they do not match, then an failure and an errorIndication (authenticationFailure) is returned to the calling module.

7) The authenticatedWholeMsg and statusInformation indicating success are then returned to the caller.

7. HMAC-SHA-96 Authentication Protocol

This section describes the HMAC-SHA-96 authentication protocol. This protocol uses the SHA hash-function which is described in [SHA-NIST], in HMAC mode described in [RFC2104], truncating the output to 96 bits.

This protocol is identified by usmHMACSHAAuthProtocol.

Over time, other authentication protocols may be defined either as a replacement of this protocol or in addition to this protocol.

7.1. Mechanisms

- In support of data integrity, a message digest algorithm is required. A digest is calculated over an appropriate portion of an SNMP message and included as part of the message sent to the recipient.

- In support of data origin authentication and data integrity, a secret value is prepended to the SNMP message prior to computing the digest; the calculated digest is then partially inserted into the message prior to transmission. The prepended secret is not transmitted. The secret value is shared by all SNMP engines authorized to originate messages on behalf of the appropriate user.

7.1.1. Digest Authentication Mechanism

The Digest Authentication Mechanism defined in this memo provides for:

- verification of the integrity of a received message, i.e., the message received is the message sent.

  The integrity of the message is protected by computing a digest over an appropriate portion of the message. The digest is computed by the originator of the message, transmitted with the message, and verified by the recipient of the message.

- verification of the user on whose behalf the message was generated.

  A secret value known only to SNMP engines authorized to generate messages on behalf of a user is used in HMAC mode (see [RFC2104]). It also recommends the hash-function output used as Message Authentication Code, to be truncated.

This mechanism uses the SHA [SHA-NIST] message digest algorithm. A 160-bit SHA digest is calculated in a special (HMAC) way over the designated portion of an SNMP message and the first 96 bits of this digest is included as part of the message sent to the recipient. The size of the digest carried in a message is 12 octets. The size of the private authentication key (the secret) is 20 octets. For the details see section 7.3.

7.2. Elements of the HMAC-SHA-96 Authentication Protocol

This section contains definitions required to realize the authentication module defined in this section of this memo.

7.2.1. Users

Authentication using this authentication protocol makes use of a defined set of userNames. For any user on whose behalf a message must be authenticated at a particular SNMP engine, that SNMP engine must have knowledge of that user. An SNMP engine that wishes to communicate with another SNMP engine must also have knowledge of a user known to that engine, including knowledge of the applicable attributes of that user.

A user and its attributes are defined as follows:

<userName>
  A string representing the name of the user.
<authKey>
  A user's secret key to be used when calculating a digest.
  It MUST be 20 octets long for SHA.

7.2.2. msgAuthoritativeEngineID

The msgAuthoritativeEngineID value contained in an authenticated message specifies the authoritative SNMP engine for that particular message (see the definition of SnmpEngineID in the SNMP Architecture document [RFC3411]).

The user's (private) authentication key is normally different at each authoritative SNMP engine and so the snmpEngineID is used to select the proper key for the authentication process.

7.2.3. SNMP Messages Using this Authentication Protocol

Messages using this authentication protocol carry a msgAuthenticationParameters field as part of the msgSecurityParameters. For this protocol, the msgAuthenticationParameters field is the serialized OCTET STRING representing the first 12 octets of HMAC-SHA-96 output done over the wholeMsg.

The digest is calculated over the wholeMsg so if a message is authenticated, that also means that all the fields in the message are intact and have not been tampered with.

7.2.4. Services Provided by the HMAC-SHA-96 Authentication Module

This section describes the inputs and outputs that the HMAC-SHA-96 Authentication module expects and produces when the User-based Security module calls the HMAC-SHA-96 Authentication module for services.

7.2.4.1. Services for Generating an Outgoing SNMP Message

HMAC-SHA-96 authentication protocol assumes that the selection of the authKey is done by the caller and that the caller passes the secret key to be used.

Upon completion the authentication module returns statusInformation and, if the message digest was correctly calculated, the wholeMsg with the digest inserted at the proper place. The abstract service primitive is:

```
statusInformation =              -- success or failure
  authenticateOutgoingMsg(
  IN   authKey                   -- secret key for authentication
  IN   wholeMsg                  -- unauthenticated complete message
  OUT  authenticatedWholeMsg     -- complete authenticated message
       )
```

The abstract data elements are:

statusInformation
  An indication of whether the authentication process was successful. If not it is an indication of the problem.

authKey
  The secret key to be used by the authentication algorithm. The length of this key MUST be 20 octets.

wholeMsg
The message to be authenticated.

authenticatedWholeMsg
The authenticated message (including inserted digest) on output.

Note, that authParameters field is filled by the authentication module and this field should be already present in the wholeMsg before the Message Authentication Code (MAC) is generated.

7.2.4.2. Services for Processing an Incoming SNMP Message

HMAC-SHA-96 authentication protocol assumes that the selection of the authKey is done by the caller and that the caller passes the secret key to be used.

Upon completion the authentication module returns statusInformation and, if the message digest was correctly calculated, the wholeMsg as it was processed. The abstract service primitive is:

```
statusInformation =              -- success or failure
  authenticateIncomingMsg(
  IN   authKey                   -- secret key for authentication
  IN   authParameters            -- as received on the wire
  IN   wholeMsg                  -- as received on the wire
  OUT  authenticatedWholeMsg     -- complete authenticated message
       )
```

The abstract data elements are:

statusInformation
An indication of whether the authentication process was successful. If not it is an indication of the problem.

authKey
The secret key to be used by the authentication algorithm. The length of this key MUST be 20 octets.

authParameters
The authParameters from the incoming message.

wholeMsg
The message to be authenticated on input and the authenticated message on output.

authenticatedWholeMsg
The whole message after the authentication check is complete.

7.3. Elements of Procedure

This section describes the procedures for the HMAC-SHA-96 authentication protocol.

7.3.1. Processing an Outgoing Message

This section describes the procedure followed by an SNMP engine whenever it must authenticate an outgoing message using the usmHMACSHAAuthProtocol.

1) The msgAuthenticationParameters field is set to the serialization, according to the rules in [RFC3417], of an OCTET STRING containing 12 zero octets.

2) From the secret authKey, two keys K1 and K2 are derived:

   a) extend the authKey to 64 octets by appending 44 zero octets; save it as extendedAuthKey

   b) obtain IPAD by replicating the octet 0x36 64 times;

   c) obtain K1 by XORing extendedAuthKey with IPAD;

   d) obtain OPAD by replicating the octet 0x5C 64 times;

   e) obtain K2 by XORing extendedAuthKey with OPAD.

3) Prepend K1 to the wholeMsg and calculate the SHA digest over it according to [SHA-NIST].

4) Prepend K2 to the result of the step 4 and calculate SHA digest over it according to [SHA-NIST]. Take the first 12 octets of the final digest - this is Message Authentication Code (MAC).

5) Replace the msgAuthenticationParameters field with MAC obtained in the step 5.

6) The authenticatedWholeMsg is then returned to the caller together with statusInformation indicating success.

7.3.2. Processing an Incoming Message

This section describes the procedure followed by an SNMP engine whenever it must authenticate an incoming message using the usmHMACSHAAuthProtocol.

1) If the digest received in the msgAuthenticationParameters field is not 12 octets long, then an failure and an errorIndication (authenticationError) is returned to the calling module.

2) The MAC received in the msgAuthenticationParameters field is saved.

3) The digest in the msgAuthenticationParameters field is replaced by the 12 zero octets.

4) From the secret authKey, two keys K1 and K2 are derived:

   a) extend the authKey to 64 octets by appending 44 zero octets; save it as extendedAuthKey

   b) obtain IPAD by replicating the octet 0x36 64 times;

   c) obtain K1 by XORing extendedAuthKey with IPAD;

   d) obtain OPAD by replicating the octet 0x5C 64 times;

   e) obtain K2 by XORing extendedAuthKey with OPAD.

5) The MAC is calculated over the wholeMsg:

   a) prepend K1 to the wholeMsg and calculate the SHA digest over it;

   b) prepend K2 to the result of step 5.a and calculate the SHA digest over it;

   c) first 12 octets of the result of step 5.b is the MAC.

   The msgAuthenticationParameters field is replaced with the MAC value that was saved in step 2.

6) The the newly calculated MAC is compared with the MAC saved in step 2. If they do not match, then a failure and an errorIndication (authenticationFailure) are returned to the calling module.

7) The authenticatedWholeMsg and statusInformation indicating success are then returned to the caller.

8. CBC-DES Symmetric Encryption Protocol

This section describes the CBC-DES Symmetric Encryption Protocol. This protocol is the first privacy protocol defined for the User-based Security Model.

This protocol is identified by usmDESPrivProtocol.

Over time, other privacy protocols may be defined either as a replacement of this protocol or in addition to this protocol.

8.1. Mechanisms

- In support of data confidentiality, an encryption algorithm is required. An appropriate portion of the message is encrypted prior to being transmitted. The User-based Security Model specifies that the scopedPDU is the portion of the message that needs to be encrypted.

- A secret value in combination with a timeliness value is used to create the en/decryption key and the initialization vector. The secret value is shared by all SNMP engines authorized to originate messages on behalf of the appropriate user.

8.1.1. Symmetric Encryption Protocol

The Symmetric Encryption Protocol defined in this memo provides support for data confidentiality. The designated portion of an SNMP message is encrypted and included as part of the message sent to the recipient.

Two organizations have published specifications defining the DES: the National Institute of Standards and Technology (NIST) [DES-NIST] and the American National Standards Institute [DES-ANSI]. There is a companion Modes of Operation specification for each definition ([DESO-NIST] and [DESO-ANSI], respectively).

The NIST has published three additional documents that implementors may find useful.

- There is a document with guidelines for implementing and using the DES, including functional specifications for the DES and its modes of operation [DESG-NIST].

- There is a specification of a validation test suite for the DES [DEST-NIST]. The suite is designed to test all aspects of the DES and is useful for pinpointing specific problems.

- There is a specification of a maintenance test for the DES [DESM-NIST]. The test utilizes a minimal amount of data and processing to test all components of the DES. It provides a simple yes-or-no indication of correct operation and is useful to run as part of an initialization step, e.g., when a computer re-boots.

8.1.1.1. DES key and Initialization Vector

The first 8 octets of the 16-octet secret (private privacy key) are used as a DES key. Since DES uses only 56 bits, the Least Significant Bit in each octet is disregarded.

The Initialization Vector for encryption is obtained using the following procedure.

The last 8 octets of the 16-octet secret (private privacy key) are used as pre-IV.

In order to ensure that the IV for two different packets encrypted by the same key, are not the same (i.e., the IV does not repeat) we need to "salt" the pre-IV with something unique per packet. An 8-octet string is used as the "salt". The concatenation of the generating SNMP engine's 32-bit snmpEngineBoots and a local 32-bit integer, that the encryption engine maintains, is input to the "salt". The 32-bit integer is initialized to an arbitrary value at boot time.

The 32-bit snmpEngineBoots is converted to the first 4 octets (Most Significant Byte first) of our "salt". The 32-bit integer is then converted to the last 4 octet (Most Significant Byte first) of our "salt". The resulting "salt" is then XOR-ed with the pre-IV to obtain the IV. The 8-octet "salt" is then put into the privParameters field encoded as an OCTET STRING. The "salt" integer is then modified. We recommend that it be incremented by one and wrap when it reaches the maximum value.

How exactly the value of the "salt" (and thus of the IV) varies, is an implementation issue, as long as the measures are taken to avoid producing a duplicate IV.

The "salt" must be placed in the privParameters field to enable the receiving entity to compute the correct IV and to decrypt the message.

8.1.1.2. Data Encryption

The data to be encrypted is treated as sequence of octets. Its length should be an integral multiple of 8 - and if it is not, the data is padded at the end as necessary. The actual pad value is irrelevant.

The data is encrypted in Cipher Block Chaining mode.

The plaintext is divided into 64-bit blocks.

The plaintext for each block is XOR-ed with the ciphertext of the previous block, the result is encrypted and the output of the encryption is the ciphertext for the block. This procedure is repeated until there are no more plaintext blocks.

For the very first block, the Initialization Vector is used instead of the ciphertext of the previous block.

8.1.1.3. Data Decryption

Before decryption, the encrypted data length is verified. If the length of the OCTET STRING to be decrypted is not an integral multiple of 8 octets, the decryption process is halted and an appropriate exception noted. When decrypting, the padding is ignored.

The first ciphertext block is decrypted, the decryption output is XOR-ed with the Initialization Vector, and the result is the first plaintext block.

For each subsequent block, the ciphertext block is decrypted, the decryption output is XOR-ed with the previous ciphertext block and the result is the plaintext block.

8.2. Elements of the DES Privacy Protocol

This section contains definitions required to realize the privacy module defined by this memo.

8.2.1. Users

Data en/decryption using this Symmetric Encryption Protocol makes use of a defined set of userNames. For any user on whose behalf a message must be en/decrypted at a particular SNMP engine, that SNMP engine must have knowledge of that user. An SNMP engine that wishes

to communicate with another SNMP engine must also have knowledge of a user known to that SNMP engine, including knowledge of the applicable attributes of that user.

A user and its attributes are defined as follows:

<userName>
An octet string representing the name of the user.

<privKey>
A user's secret key to be used as input for the DES key and IV. The length of this key MUST be 16 octets.

8.2.2. msgAuthoritativeEngineID

The msgAuthoritativeEngineID value contained in an authenticated message specifies the authoritative SNMP engine for that particular message (see the definition of SnmpEngineID in the SNMP Architecture document [RFC3411]).

The user's (private) privacy key is normally different at each authoritative SNMP engine and so the snmpEngineID is used to select the proper key for the en/decryption process.

8.2.3. SNMP Messages Using this Privacy Protocol

Messages using this privacy protocol carry a msgPrivacyParameters field as part of the msgSecurityParameters. For this protocol, the msgPrivacyParameters field is the serialized OCTET STRING representing the "salt" that was used to create the IV.

8.2.4. Services Provided by the DES Privacy Module

This section describes the inputs and outputs that the DES Privacy module expects and produces when the User-based Security module invokes the DES Privacy module for services.

8.2.4.1. Services for Encrypting Outgoing Data

This DES privacy protocol assumes that the selection of the privKey is done by the caller and that the caller passes the secret key to be used.

Upon completion the privacy module returns statusInformation and, if the encryption process was successful, the encryptedPDU and the msgPrivacyParameters encoded as an OCTET STRING. The abstract service primitive is:

```
statusInformation =              -- success of failure
  encryptData(
  IN    encryptKey               -- secret key for encryption
  IN    dataToEncrypt            -- data to encrypt (scopedPDU)
  OUT   encryptedData            -- encrypted data (encryptedPDU)
  OUT   privParameters           -- filled in by service provider
        )
```

The abstract data elements are:

statusInformation
An indication of the success or failure of the encryption process. In case of failure, it is an indication of the error.

encryptKey
The secret key to be used by the encryption algorithm. The length of this key MUST be 16 octets.

dataToEncrypt
The data that must be encrypted.

encryptedData
The encrypted data upon successful completion.

privParameters
The privParameters encoded as an OCTET STRING.

8.2.4.2. Services for Decrypting Incoming Data

This DES privacy protocol assumes that the selection of the privKey is done by the caller and that the caller passes the secret key to be used.

Upon completion the privacy module returns statusInformation and, if the decryption process was successful, the scopedPDU in plain text. The abstract service primitive is:

```
statusInformation =
  decryptData(
  IN    decryptKey               -- secret key for decryption
  IN    privParameters           -- as received on the wire
  IN    encryptedData            -- encrypted data (encryptedPDU)
  OUT   decryptedData            -- decrypted data (scopedPDU)
        )
```

The abstract data elements are:

statusInformation
An indication whether the data was successfully decrypted and if not an indication of the error.

decryptKey
The secret key to be used by the decryption algorithm. The length of this key MUST be 16 octets.

privParameters
The "salt" to be used to calculate the IV.

encryptedData
The data to be decrypted.

decryptedData
The decrypted data.

8.3. Elements of Procedure.

This section describes the procedures for the DES privacy protocol.

8.3.1. Processing an Outgoing Message

This section describes the procedure followed by an SNMP engine whenever it must encrypt part of an outgoing message using the usmDESPrivProtocol.

1) The secret cryptKey is used to construct the DES encryption key, the "salt" and the DES pre-IV (from which the IV is computed as described in section 8.1.1.1).

2) The privParameters field is set to the serialization according to the rules in [RFC3417] of an OCTET STRING representing the "salt" string.

3) The scopedPDU is encrypted (as described in section 8.1.1.2) and the encrypted data is serialized according to the rules in [RFC3417] as an OCTET STRING.

4) The serialized OCTET STRING representing the encrypted scopedPDU together with the privParameters and statusInformation indicating success is returned to the calling module.

8.3.2. Processing an Incoming Message

This section describes the procedure followed by an SNMP engine whenever it must decrypt part of an incoming message using the usmDESPrivProtocol.

1) If the privParameters field is not an 8-octet OCTET STRING, then an error indication (decryptionError) is returned to the calling module.

2) The "salt" is extracted from the privParameters field.

3) The secret cryptKey and the "salt" are then used to construct the DES decryption key and pre-IV (from which the IV is computed as described in section 8.1.1.1).

4) The encryptedPDU is then decrypted (as described in section 8.1.1.3).

5) If the encryptedPDU cannot be decrypted, then an error indication (decryptionError) is returned to the calling module.

6) The decrypted scopedPDU and statusInformation indicating success are returned to the calling module.

9. Intellectual Property

10. Acknowledgements

This document is the result of the efforts of the SNMPv3 Working Group. Some special thanks are in order to the following SNMPv3 WG members:

Harald Tveit Alvestrand (Maxware)
Dave Battle (SNMP Research, Inc.)
Alan Beard (Disney Worldwide Services)
Paul Berrevoets (SWI Systemware/Halcyon Inc.)
Martin Bjorklund (Ericsson)
Uri Blumenthal (IBM T.J. Watson Research Center)
Jeff Case (SNMP Research, Inc.)
John Curran (BBN)
Mike Daniele (Compaq Computer Corporation))
T. Max Devlin (Eltrax Systems)
John Flick (Hewlett Packard)
Rob Frye (MCI)
Wes Hardaker (U.C.Davis, Information Technology - D.C.A.S.)
David Harrington (Cabletron Systems Inc.)
Lauren Heintz (BMC Software, Inc.)
N.C. Hien (IBM T.J. Watson Research Center)
Michael Kirkham (InterWorking Labs, Inc.)
Dave Levi (SNMP Research, Inc.)
Louis A Mamakos (UUNET Technologies Inc.)
Joe Marzot (Nortel Networks)
Paul Meyer (Secure Computing Corporation)
Keith McCloghrie (Cisco Systems)
Bob Moore (IBM)
Russ Mundy (TIS Labs at Network Associates)
Bob Natale (ACE*COMM Corporation)
Mike O'Dell (UUNET Technologies Inc.)
Dave Perkins (DeskTalk)
Peter Polkinghorne (Brunel University)
Randy Presuhn (BMC Software, Inc.)
David Reeder (TIS Labs at Network Associates)
David Reid (SNMP Research, Inc.)
Aleksey Romanov (Quality Quorum)
Shawn Routhier (Epilogue)
Juergen Schoenwaelder (TU Braunschweig)
Bob Stewart (Cisco Systems)
Mike Thatcher (Independent Consultant)
Bert Wijnen (IBM T.J. Watson Research Center)

The document is based on recommendations of the IETF Security and Administrative Framework Evolution for SNMP Advisory Team. Members of that Advisory Team were:

David Harrington (Cabletron Systems Inc.)
Jeff Johnson (Cisco Systems)
David Levi (SNMP Research Inc.)
John Linn (Openvision)
Russ Mundy (Trusted Information Systems) chair
Shawn Routhier (Epilogue)
Glenn Waters (Nortel)
Bert Wijnen (IBM T. J. Watson Research Center)

As recommended by the Advisory Team and the SNMPv3 Working Group Charter, the design incorporates as much as practical from previous RFCs and drafts. As a result, special thanks are due to the authors of previous designs known as SNMPv2u and SNMPv2*:

Jeff Case (SNMP Research, Inc.)
David Harrington (Cabletron Systems Inc.)
David Levi (SNMP Research, Inc.)
Keith McCloghrie (Cisco Systems)
Brian O'Keefe (Hewlett Packard)
Marshall T. Rose (Dover Beach Consulting)
Jon Saperia (BGS Systems Inc.)
Steve Waldbusser (International Network Services)
Glenn W. Waters (Bell-Northern Research Ltd.)

11. Security Considerations

11.1. Recommended Practices

This section describes practices that contribute to the secure, effective operation of the mechanisms defined in this memo.

- An SNMP engine must discard SNMP Response messages that do not correspond to any currently outstanding Request message. It is the responsibility of the Message Processing module to take care of this. For example it can use a msgID for that.

  An SNMP Command Generator Application must discard any Response Class PDU for which there is no currently outstanding Confirmed Class PDU; for example for SNMPv2 [RFC3416] PDUs, the request-id component in the PDU can be used to correlate Responses to outstanding Requests.

Although it would be typical for an SNMP engine and an SNMP Command Generator Application to do this as a matter of course, when using these security protocols it is significant due to the possibility of message duplication (malicious or otherwise).

- If an SNMP engine uses a msgID for correlating Response messages to outstanding Request messages, then it MUST use different msgIDs in all such Request messages that it sends out during a Time Window (150 seconds) period.

  A Command Generator or Notification Originator Application MUST use different request-ids in all Request PDUs that it sends out during a TimeWindow (150 seconds) period.

  This must be done to protect against the possibility of message duplication (malicious or otherwise).

  For example, starting operations with a msgID and/or request-id value of zero is not a good idea. Initializing them with an unpredictable number (so they do not start out the same after each reboot) and then incrementing by one would be acceptable.

- An SNMP engine should perform time synchronization using authenticated messages in order to protect against the possibility of message duplication (malicious or otherwise).

- When sending state altering messages to a managed authoritative SNMP engine, a Command Generator Application should delay sending successive messages to that managed SNMP engine until a positive acknowledgement is received for the previous message or until the previous message expires.

  No message ordering is imposed by the SNMP. Messages may be received in any order relative to their time of generation and each will be processed in the ordered received. Note that when an authenticated message is sent to a managed SNMP engine, it will be valid for a period of time of approximately 150 seconds under normal circumstances, and is subject to replay during this period. Indeed, an SNMP engine and SNMP Command Generator Applications must cope with the loss and re-ordering of messages resulting from anomalies in the network as a matter of course.

  However, a managed object, snmpSetSerialNo [RFC3418], is specifically defined for use with SNMP Set operations in order to provide a mechanism to ensure that the processing of SNMP messages occurs in a specific order.

- The frequency with which the secrets of a User-based Security Model user should be changed is indirectly related to the frequency of their use.

  Protecting the secrets from disclosure is critical to the overall security of the protocols. Frequent use of a secret provides a continued source of data that may be useful to a cryptanalyst in exploiting known or perceived weaknesses in an algorithm. Frequent changes to the secret avoid this vulnerability.

  Changing a secret after each use is generally regarded as the most secure practice, but a significant amount of overhead may be associated with that approach.

  Note, too, in a local environment the threat of disclosure may be less significant, and as such the changing of secrets may be less frequent. However, when public data networks are used as the communication paths, more caution is prudent.

11.2 Defining Users

The mechanisms defined in this document employ the notion of users on whose behalf messages are sent. How "users" are defined is subject to the security policy of the network administration. For example, users could be individuals (e.g., "joe" or "jane"), or a particular role (e.g., "operator" or "administrator"), or a combination (e.g., "joe-operator", "jane-operator" or "joe-admin"). Furthermore, a user may be a logical entity, such as an SNMP Application or a set of SNMP Applications, acting on behalf of an individual or role, or set of individuals, or set of roles, including combinations.

Appendix A describes an algorithm for mapping a user "password" to a 16/20 octet value for use as either a user's authentication key or privacy key (or both). Note however, that using the same password (and therefore the same key) for both authentication and privacy is very poor security practice and should be strongly discouraged. Passwords are often generated, remembered, and input by a human. Human-generated passwords may be less than the 16/20 octets required by the authentication and privacy protocols, and brute force attacks can be quite easy on a relatively short ASCII character set. Therefore, the algorithm is Appendix A performs a transformation on the password. If the Appendix A algorithm is used, SNMP implementations (and SNMP configuration applications) must ensure that passwords are at least 8 characters in length. Please note that longer passwords with repetitive strings may result in exactly the same key. For example, a password 'bertbert' will result in exactly the same key as password 'bertbertbert'.

Because the Appendix A algorithm uses such passwords (nearly) directly, it is very important that they not be easily guessed. It is suggested that they be composed of mixed-case alphanumeric and punctuation characters that don't form words or phrases that might be found in a dictionary. Longer passwords improve the security of the system. Users may wish to input multiword phrases to make their password string longer while ensuring that it is memorable.

Since it is infeasible for human users to maintain different passwords for every SNMP engine, but security requirements strongly discourage having the same key for more than one SNMP engine, the User-based Security Model employs a compromise proposed in [Localized-key]. It derives the user keys for the SNMP engines from user's password in such a way that it is practically impossible to either determine the user's password, or user's key for another SNMP engine from any combination of user's keys on SNMP engines.

Note however, that if user's password is disclosed, then key localization will not help and network security may be compromised in this case. Therefore a user's password or non-localized key MUST NOT be stored on a managed device/node. Instead the localized key SHALL be stored (if at all), so that, in case a device does get compromised, no other managed or managing devices get compromised.

11.3. Conformance

To be termed a "Secure SNMP implementation" based on the User-based Security Model, an SNMP implementation MUST:

- implement one or more Authentication Protocol(s). The HMAC-MD5-96 and HMAC-SHA-96 Authentication Protocols defined in this memo are examples of such protocols.

- to the maximum extent possible, prohibit access to the secret(s) of each user about which it maintains information in a Local Configuration Datastore (LCD) under all circumstances except as required to generate and/or validate SNMP messages with respect to that user.

- implement the key-localization mechanism.

- implement the SNMP-USER-BASED-SM-MIB.

In addition, an authoritative SNMP engine SHOULD provide initial configuration in accordance with Appendix A.1.

Implementation of a Privacy Protocol (the DES Symmetric Encryption Protocol defined in this memo is one such protocol) is optional.

11.4. Use of Reports

The use of unsecure reports (i.e., sending them with a securityLevel of noAuthNoPriv) potentially exposes a non-authoritative SNMP engine to some form of attacks. Some people consider these denial of service attacks, others don't. An installation should evaluate the risk involved before deploying unsecure Report PDUs.

11.5 Access to the SNMP-USER-BASED-SM-MIB

The objects in this MIB may be considered sensitive in many environments. Specifically the objects in the usmUserTable contain information about users and their authentication and privacy protocols. It is important to closely control (both read and write) access to these MIB objects by using appropriately configured Access Control models (for example the View-based Access Control Model as specified in [RFC3415]).

12. References

12.1 Normative References

[RFC1321] Rivest, R., "Message Digest Algorithm MD5", RFC 1321, April 1992.

[RFC2104] Krawczyk, H., Bellare, M. and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2578] McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M. and S. Waldbusser, "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, April 1999.

[RFC2579] McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M. and S. Waldbusser, "Textual Conventions for SMIv2", STD 58, RFC 2579, April 1999.

[RFC2580] McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M. and S. Waldbusser, "Conformance Statements for SMIv2", STD 58, RFC 2580, April 1999.

[RFC3411] Harrington, D., Presuhn, R. and B. Wijnen, "An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks", STD 62, RFC 3411, December 2002.

[RFC3412] 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.

[RFC3415] 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.

[RFC3416] Presuhn, R., Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3416, December 2002.

[RFC3417] Presuhn, R., Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Transport Mappings for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3417, December 2002.

[RFC3418] Presuhn, R., Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Management Information Base (MIB) for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3418, December 2002.

[DES-NIST] Data Encryption Standard, National Institute of Standards and Technology. Federal Information Processing Standard (FIPS) Publication 46-1. Supersedes FIPS Publication 46, (January, 1977; reaffirmed January, 1988).

[DESO-NIST] DES Modes of Operation, National Institute of Standards and Technology. Federal Information Processing Standard (FIPS) Publication 81, (December, 1980).

[SHA-NIST] Secure Hash Algorithm. NIST FIPS 180-1, (April, 1995) http://csrc.nist.gov/fips/fip180-1.txt (ASCII) http://csrc.nist.gov/fips/fip180-1.ps (Postscript)

12.1 Informative References

[Localized-Key] U. Blumenthal, N. C. Hien, B. Wijnen "Key Derivation for Network Management Applications" IEEE Network Magazine, April/May issue, 1997.

[DES-ANSI] Data Encryption Algorithm, American National Standards Institute. ANSI X3.92-1981, (December, 1980).

[DESO-ANSI] Data Encryption Algorithm - Modes of Operation, American National Standards Institute. ANSI X3.106-1983, (May 1983).

[DESG-NIST] Guidelines for Implementing and Using the NBS Data Encryption Standard, National Institute of Standards and Technology. Federal Information Processing Standard (FIPS) Publication 74, (April, 1981).

[DEST-NIST] Validating the Correctness of Hardware Implementations of the NBS Data Encryption Standard, National Institute of Standards and Technology. Special Publication 500-20.

[DESM-NIST] Maintenance Testing for the Data Encryption Standard, National Institute of Standards and Technology. Special Publication 500-61, (August, 1980).

[RFC3174] Eastlake, D. 3rd and P. Jones, "US Secure Hash Algorithm 1 (SHA1)", RFC 3174, September 2001.

APPENDIX A - Installation

A.1. SNMP engine Installation Parameters

During installation, an authoritative SNMP engine SHOULD (in the meaning as defined in [RFC2119]) be configured with several initial parameters. These include:

1) A Security Posture

The choice of security posture determines if initial configuration is implemented and if so how. One of three possible choices is selected:

minimum-secure,
semi-secure,
very-secure (i.e., no-initial-configuration)

In the case of a very-secure posture, there is no initial configuration, and so the following steps are irrelevant.

2) One or More Secrets

These are the authentication/privacy secrets for the first user to be configured.

One way to accomplish this is to have the installer enter a "password" for each required secret. The password is then algorithmically converted into the required secret by:

- forming a string of length 1,048,576 octets by repeating the value of the password as often as necessary, truncating accordingly, and using the resulting string as the input to the MD5 algorithm [RFC1321]. The resulting digest, termed "digest1", is used in the next step.

- a second string is formed by concatenating digest1, the SNMP engine's snmpEngineID value, and digest1. This string is used as input to the MD5 algorithm [RFC1321].

  The resulting digest is the required secret (see Appendix A.2).

With these configured parameters, the SNMP engine instantiates the following usmUserEntry in the usmUserTable:

	no privacy support	privacy support
usmUserEngineID	localEngineID	localEngineID
usmUserName	"initial"	"initial"
usmUserSecurityName	"initial"	"initial"
usmUserCloneFrom	ZeroDotZero	ZeroDotZero
usmUserAuthProtocol	usmHMACMD5AuthProtocol	usmHMACMD5AuthProtocol
usmUserAuthKeyChange	""	""
usmUserOwnAuthKeyChange	""	""
usmUserPrivProtocol	none	usmDESPrivProtocol
usmUserPrivKeyChange	""	""
usmUserOwnPrivKeyChange	""	""
usmUserPublic	""	""
usmUserStorageType	anyValidStorageType	anyValidStorageType
usmUserStatus	active	active

It is recommended to also instantiate a set of template usmUserEntries which can be used as clone-from users for newly created usmUserEntries. These are the two suggested entries:

	no privacy support	privacy support
usmUserEngineID	localEngineID	localEngineID
usmUserName	"templateMD5"	"templateMD5"
usmUserSecurityName	"templateMD5"	"templateMD5"
usmUserCloneFrom	ZeroDotZero	ZeroDotZero
usmUserAuthProtocol	usmHMACMD5AuthProtocol	usmHMACMD5AuthProtocol
usmUserAuthKeyChange	""	""
usmUserOwnAuthKeyChange	""	""
usmUserPrivProtocol	none	usmDESPrivProtocol
usmUserPrivKeyChange	""	""
usmUserOwnPrivKeyChange	""	""
usmUserPublic	""	""
usmUserStorageType	permanent	permanent
usmUserStatus	active	active

	no privacy support	privacy support
usmUserEngineID	localEngineID	localEngineID
usmUserName	"templateSHA"	"templateSHA"
usmUserSecurityName	"templateSHA"	"templateSHA"
usmUserCloneFrom	ZeroDotZero	ZeroDotZero
usmUserAuthProtocol	usmHMACSHAAuthProtocol	usmHMACSHAAuthProtocol
usmUserAuthKeyChange	""	""
usmUserOwnAuthKeyChange	""	""
usmUserPrivProtocol	none	usmDESPrivProtocol
usmUserPrivKeyChange	""	""
usmUserOwnPrivKeyChange	""	""
usmUserPublic	""	""
usmUserStorageType	permanent	permanent
usmUserStatus	active	active

A.2. Password to Key Algorithm

A sample code fragment (section A.2.1) demonstrates the password to key algorithm which can be used when mapping a password to an authentication or privacy key using MD5. The reference source code of MD5 is available in [RFC1321].

Another sample code fragment (section A.2.2) demonstrates the password to key algorithm which can be used when mapping a password to an authentication or privacy key using SHA (documented in SHA-NIST).

An example of the results of a correct implementation is provided (section A.3) which an implementor can use to check if his implementation produces the same result.

A.2.1. Password to Key Sample Code for MD5

```
   void password_to_key_md5(
      u_char *password,    /* IN */
      u_int   passwordlen, /* IN */
      u_char *engineID,    /* IN  - pointer to snmpEngineID  */
      u_int   engineLength,/* IN  - length of snmpEngineID */
      u_char *key)         /* OUT - pointer to caller 16-octet buffer */
   {
      MD5_CTX     MD;
      u_char     *cp, password_buf[64];
      u_long      password_index = 0;
      u_long      count = 0, i;

      MD5Init (&MD);   /* initialize MD5 */

      /**********************************************/
      /* Use while loop until we've done 1 Megabyte */
      /**********************************************/
      while (count < 1048576) {
         cp = password_buf;
         for (i = 0; i < 64; i++) {
             /*************************************************/
             /* Take the next octet of the password, wrapping */
             /* to the beginning of the password as necessary.*/
             /*************************************************/
             *cp++ = password[password_index++ % passwordlen];
         }
         MD5Update (&MD, password_buf, 64);
         count += 64;
      }
      MD5Final (key, &MD);          /* tell MD5 we're done */

      /*****************************************************/
      /* Now localize the key with the engineID and pass   */
      /* through MD5 to produce final key                  */
      /* May want to ensure that engineLength <= 32,       */
      /* otherwise need to use a buffer larger than 64     */
      /*****************************************************/
      memcpy(password_buf, key, 16);
      memcpy(password_buf+16, engineID, engineLength);
      memcpy(password_buf+16+engineLength, key, 16);

      MD5Init(&MD);
      MD5Update(&MD, password_buf, 32+engineLength);
      MD5Final(key, &MD);
      return;
   }
```

A.2.2. Password to Key Sample Code for SHA

```
   void password_to_key_sha(
      u_char *password,    /* IN */
      u_int   passwordlen, /* IN */
      u_char *engineID,    /* IN  - pointer to snmpEngineID  */
      u_int   engineLength,/* IN  - length of snmpEngineID */
      u_char *key)         /* OUT - pointer to caller 20-octet buffer */
   {
      SHA_CTX     SH;
      u_char     *cp, password_buf[72];
      u_long      password_index = 0;
      u_long      count = 0, i;

      SHAInit (&SH);   /* initialize SHA */

      /**********************************************/
      /* Use while loop until we've done 1 Megabyte */
      /**********************************************/
      while (count < 1048576) {
         cp = password_buf;
         for (i = 0; i < 64; i++) {
             /*************************************************/
             /* Take the next octet of the password, wrapping */
             /* to the beginning of the password as necessary.*/
             /*************************************************/
             *cp++ = password[password_index++ % passwordlen];
         }
         SHAUpdate (&SH, password_buf, 64);
         count += 64;
      }
      SHAFinal (key, &SH);          /* tell SHA we're done */

      /*****************************************************/
      /* Now localize the key with the engineID and pass   */
      /* through SHA to produce final key                  */
      /* May want to ensure that engineLength <= 32,       */
      /* otherwise need to use a buffer larger than 72     */
      /*****************************************************/
      memcpy(password_buf, key, 20);
      memcpy(password_buf+20, engineID, engineLength);
      memcpy(password_buf+20+engineLength, key, 20);

      SHAInit(&SH);
      SHAUpdate(&SH, password_buf, 40+engineLength);
      SHAFinal(key, &SH);
      return;
   }
```

A.3. Password to Key Sample Results

A.3.1. Password to Key Sample Results using MD5

The following shows a sample output of the password to key algorithm for a 16-octet key using MD5.

With a password of "maplesyrup" the output of the password to key algorithm before the key is localized with the SNMP engine's snmpEngineID is:

'9f af 32 83 88 4e 92 83 4e bc 98 47 d8 ed d9 63'H

After the intermediate key (shown above) is localized with the snmpEngineID value of:

'00 00 00 00 00 00 00 00 00 00 00 02'H

the final output of the password to key algorithm is:

'52 6f 5e ed 9f cc e2 6f 89 64 c2 93 07 87 d8 2b'H

A.3.2. Password to Key Sample Results using SHA

The following shows a sample output of the password to key algorithm for a 20-octet key using SHA.

With a password of "maplesyrup" the output of the password to key algorithm before the key is localized with the SNMP engine's snmpEngineID is:

'9f b5 cc 03 81 49 7b 37 93 52 89 39 ff 78 8d 5d 79 14 52 11'H

After the intermediate key (shown above) is localized with the snmpEngineID value of:

'00 00 00 00 00 00 00 00 00 00 00 02'H

the final output of the password to key algorithm is:

'66 95 fe bc 92 88 e3 62 82 23 5f c7 15 1f 12 84 97 b3 8f 3f'H

A.4. Sample Encoding of msgSecurityParameters

The msgSecurityParameters in an SNMP message are represented as an OCTET STRING. This OCTET STRING should be considered opaque outside a specific Security Model.

The User-based Security Model defines the contents of the OCTET STRING as a SEQUENCE (see section 2.4).

Given these two properties, the following is an example of they msgSecurityParameters for the User-based Security Model, encoded as an OCTET STRING:

```
04 <length>
30 <length>
04 <length> <msgAuthoritativeEngineID>
02 <length> <msgAuthoritativeEngineBoots>
02 <length> <msgAuthoritativeEngineTime>
04 <length> <msgUserName>
04 0c       <HMAC-MD5-96-digest>
04 08       <salt>
```

Here is the example once more, but now with real values (except for the digest in msgAuthenticationParameters and the salt in msgPrivacyParameters, which depend on variable data that we have not defined here):

Hex Data	Description	
04 39	OCTET STRING,	length 57
30 37	SEQUENCE,	length 55
04 0c 80000002	msgAuthoritativeEngineID:	IBM
01		IPv4 address
09840301		9.132.3.1
02 01 01	msgAuthoritativeEngineBoots:	1
02 02 0101	msgAuthoritativeEngineTime:	257
04 04 62657274	msgUserName:	bert
04 0c 01234567	msgAuthenticationParameters:	sample value
89abcdef		
fedcba98		
04 08 01234567	msgPrivacyParameters:	sample value
89abcdef		

A.5. Sample keyChange Results

A.5.1. Sample keyChange Results using MD5

Let us assume that a user has a current password of "maplesyrup" as in section A.3.1. and let us also assume the snmpEngineID of 12 octets:

’00 00 00 00 00 00 00 00 00 00 00 02’H

If we now want to change the password to "newsyrup", then we first calculate the key for the new password. It is as follows:

'01 ad d2 73 10 7c 4e 59 6b 4b 00 f8 2b 1d 42 a7'H

If we localize it for the above snmpEngineID, then the localized new key becomes:

'87 02 1d 7b d9 d1 01 ba 05 ea 6e 3b f9 d9 bd 4a'H

If we then use a (not so good, but easy to test) random value of:

'00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'H

Then the value we must send for keyChange is:

'00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
88 05 61 51 41 67 6c c9 19 61 74 e7 42 a3 25 51'H

If this were for the privacy key, then it would be exactly the same.

A.5.2. Sample keyChange Results using SHA

Let us assume that a user has a current password of "maplesyrup" as in section A.3.2. and let us also assume the snmpEngineID of 12 octets:

'00 00 00 00 00 00 00 00 00 00 00 02'H

If we now want to change the password to "newsyrup", then we first calculate the key for the new password. It is as follows:

'3a 51 a6 d7 36 aa 34 7b 83 dc 4a 87 e3 e5 5e e4 d6 98 ac 71'H

If we localize it for the above snmpEngineID, then the localized new key becomes:

'78 e2 dc ce 79 d5 94 03 b5 8c 1b ba a5 bf f4 63 91 f1 cd 25'H

If we then use a (not so good, but easy to test) random value of:

'00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'H

Then the value we must send for keyChange is:

'00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9c 10 17 f4 fd 48 3d 2d e8 d5 fa db f8 43 92 cb 06 45 70 51'

For the key used for privacy, the new nonlocalized key would be:

'3a 51 a6 d7 36 aa 34 7b 83 dc 4a 87 e3 e5 5e e4 d6 98 ac 71'H

For the key used for privacy, the new localized key would be (note that they localized key gets truncated to 16 octets for DES):

'78 e2 dc ce 79 d5 94 03 b5 8c 1b ba a5 bf f4 63'H

If we then use a (not so good, but easy to test) random value of:

'00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'H

Then the value we must send for keyChange for the privacy key is:

'00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
'7e f8 d8 a4 c9 cd b2 6b 47 59 1c d8 52 ff 88 b5'H

B. Change Log

Changes made since RFC2574:

- Updated references
- Updated contact info
- Clarifications
  - to first constraint item 1) on page 6.
  - to usmUserCloneFrom DESCRIPTION clause
  - to securityName in section 2.1
- Fixed "command responder" into "command generator" in last para of DESCRIPTION clause of usmUserTable.

Changes made since RFC2274:

- Fixed msgUserName to allow size of zero and explain that this can be used for snmpEngineID discovery.
- Clarified section 3.1 steps 4.b, 5, 6 and 8.b.
- Clarified section 3.2 paragraph 2.
- Clarified section 3.2 step 7.a last paragraph, step 7.b.1 second bullet and step 7.b.2 third bullet.
- Clarified section 4 to indicate that discovery can use a userName of zero length in unAuthenticated messages, whereas a valid userName must be used in authenticated messages.
- Added REVISION clauses to MODULE-IDENTITY
- Clarified KeyChange TC by adding a note that localized keys must be used when calculating a KeyChange value.
- Added clarifying text to the DESCRIPTION clause of usmUserTable. Added text describes a recommended procedure for adding a new user.
- Clarified the use of usmUserCloneFrom object.

- Clarified how and under which conditions the usmUserAuthProtocol and usmUserPrivProtocol can be initialized and/or changed.
- Added comment on typical sizes for usmUserAuthKeyChange and usmUserPrivKeyChange. Also for usmUserOwnAuthKeyChange and usmUserOwnPrivKeyChange.
- Added clarifications to the DESCRIPTION clauses of usmUserAuthKeyChange, usmUserOwnAuthKeychange, usmUserPrivKeyChange and usmUserOwnPrivKeychange.
- Added clarification to DESCRIPTION clause of usmUserStorageType.
- Added clarification to DESCRIPTION clause of usmUserStatus.
- Clarified IV generation procedure in section 8.1.1.1 and in addition clarified section 8.3.1 step 1 and section 8.3.2. step 3.
- Clarified section 11.2 and added a warning that different size passwords with repetitive strings may result in same key.
- Added template users to appendix A for cloning process.
- Fixed C-code examples in Appendix A.
- Fixed examples of generated keys in Appendix A.
- Added examples of KeyChange values to Appendix A.
- Used PDU Classes instead of RFC1905 PDU types.
- Added text in the security section about Reports and Access Control to the MIB.
- Removed a incorrect note at the end of section 3.2 step 7.
- Added a note in section 3.2 step 3.
- Corrected various spelling errors and typos.
- Corrected procedure for 3.2 step 2.a)
- various clarifications.
- Fixed references to new/revised documents
- Change to no longer cache data that is not used

Editors' Addresses

Uri Blumenthal
Lucent Technologies
67 Whippany Rd.
Whippany, NJ 07981
USA

Phone: +1-973-386-2163
EMail: uri@lucent.com

Bert Wijnen
Lucent Technologies
Schagen 33
3461 GL Linschoten
Netherlands

Phone: +31-348-480-685
EMail: bwijnen@lucent.com

Full Copyright Statement

Acknowledgement

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

Network Working Group
Request for Comments: 3415
STD: 62
Obsoletes: 2575
Category: Standards Track

B. Wijnen
Lucent Technologies
R. Presuhn
BMC Software, Inc.
K. McCloghrie
Cisco Systems, Inc.

December 2002

View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)

Status of this Memo

Copyright Notice

Abstract

This document describes the View-based Access Control Model (VACM) for use in the Simple Network Management Protocol (SNMP) architecture. It defines the Elements of Procedure for controlling access to management information. This document also includes a Management Information Base (MIB) for remotely managing the configuration parameters for the View-based Access Control Model. This document obsoletes RFC 2575.

Table of Contents

1. Introduction

The Architecture for describing Internet Management Frameworks [RFC3411] describes that an SNMP engine is composed of:

1) a Dispatcher
2) a Message Processing Subsystem,
3) a Security Subsystem, and
4) an Access Control Subsystem.

Applications make use of the services of these subsystems.

It is important to understand the SNMP architecture and its terminology to understand where the View-based Access Control Model described in this document fits into the architecture and interacts with other subsystems within the architecture. The reader is expected to have read and understood the description and terminology of the SNMP architecture, as defined in [RFC3411].

The Access Control Subsystem of an SNMP engine has the responsibility for checking whether a specific type of access (read, write, notify) to a particular object (instance) is allowed.

It is the purpose of this document to define a specific model of the Access Control Subsystem, designated the View-based Access Control Model. Note that this is not necessarily the only Access Control Model.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, RFC 2119.

1.2. Access Control

Access Control occurs (either implicitly or explicitly) in an SNMP entity when processing SNMP retrieval or modification request messages from an SNMP entity. For example a Command Responder application applies Access Control when processing requests that it received from a Command Generator application. These requests contain Read Class and Write Class PDUs as defined in [RFC3411].

Access Control also occurs in an SNMP entity when an SNMP notification message is generated (by a Notification Originator application). These notification messages contain Notification Class PDUs as defined in [RFC3411].

The View-based Access Control Model defines a set of services that an application (such as a Command Responder or a Notification Originator application) can use for checking access rights. It is the responsibility of the application to make the proper service calls for access checking.

1.3. Local Configuration Datastore

To implement the model described in this document, an SNMP entity needs to retain information about access rights and policies. This information is part of the SNMP engine's Local Configuration Datastore (LCD). See [RFC3411] for the definition of LCD.

In order to allow an SNMP entity's LCD to be remotely configured, portions of the LCD need to be accessible as managed objects. A MIB module, the View-based Access Control Model Configuration MIB, which defines these managed object types is included in this document.

2. Elements of the Model

This section contains definitions to realize the access control service provided by the View-based Access Control Model.

2.1. Groups

A group is a set of zero or more <securityModel, securityName> tuples on whose behalf SNMP management objects can be accessed. A group defines the access rights afforded to all securityNames which belong to that group. The combination of a securityModel and a securityName maps to at most one group. A group is identified by a groupName.

The Access Control module assumes that the securityName has already been authenticated as needed and provides no further authentication of its own.

The View-based Access Control Model uses the securityModel and the securityName as inputs to the Access Control module when called to check for access rights. It determines the groupName as a function of securityModel and securityName.

2.2. securityLevel

Different access rights for members of a group can be defined for different levels of security, i.e., noAuthNoPriv, authNoPriv, and authPriv. The securityLevel identifies the level of security that will be assumed when checking for access rights. See the SNMP Architecture document [RFC3411] for a definition of securityLevel.

The View-based Access Control Model requires that the securityLevel is passed as input to the Access Control module when called to check for access rights.

2.3. Contexts

An SNMP context is a collection of management information accessible by an SNMP entity. An item of management information may exist in more than one context. An SNMP entity potentially has access to many contexts. Details about the naming of management information can be found in the SNMP Architecture document [RFC3411].

The View-based Access Control Model defines a vacmContextTable that lists the locally available contexts by contextName.

2.4. MIB Views and View Families

For security reasons, it is often valuable to be able to restrict the access rights of some groups to only a subset of the management information in the management domain. To provide this capability, access to a context is via a "MIB view" which details a specific set of managed object types (and optionally, the specific instances of object types) within that context. For example, for a given context, there will typically always be one MIB view which provides access to all management information in that context, and often there will be other MIB views each of which contains some subset of the information. So, the access allowed for a group can be restricted in the desired manner by specifying its rights in terms of the particular (subset) MIB view it can access within each appropriate context.

Since managed object types (and their instances) are identified via the tree-like naming structure of ISO's OBJECT IDENTIFIERs [ISO-ASN.1, RFC2578], it is convenient to define a MIB view as the combination of a set of "view subtrees", where each view subtree is a subtree within the managed object naming tree. Thus, a simple MIB view (e.g., all managed objects within the Internet Network Management Framework) can be defined as a single view subtree, while more complicated MIB views (e.g., all information relevant to a particular network interface) can be represented by the union of multiple view subtrees.

While any set of managed objects can be described by the union of some number of view subtrees, situations can arise that would require a very large number of view subtrees. This could happen, for example, when specifying all columns in one conceptual row of a MIB table because they would appear in separate subtrees, one per column, each with a very similar format. Because the formats are similar, the required set of subtrees can easily be aggregated into one structure. This structure is named a family of view subtrees after the set of subtrees that it conceptually represents. A family of view subtrees can either be included or excluded from a MIB view.

2.4.1. View Subtree

A view subtree is the set of all MIB object instances which have a common ASN.1 OBJECT IDENTIFIER prefix to their names. A view subtree is identified by the OBJECT IDENTIFIER value which is the longest OBJECT IDENTIFIER prefix common to all (potential) MIB object instances in that subtree.

2.4.2. ViewTreeFamily

A family of view subtrees is a pairing of an OBJECT IDENTIFIER value (called the family name) together with a bit string value (called the family mask). The family mask indicates which sub-identifiers of the associated family name are significant to the family's definition.

For each possible managed object instance, that instance belongs to a particular ViewTreeFamily if both of the following conditions are true:

- the OBJECT IDENTIFIER name of the managed object instance contains at least as many sub-identifiers as does the family name, and

- each sub-identifier in the OBJECT IDENTIFIER name of the managed object instance matches the corresponding sub-identifier of the family name whenever the corresponding bit of the associated family mask is non-zero.

When the configured value of the family mask is all ones, the view subtree family is identical to the single view subtree identified by the family name.

When the configured value of the family mask is shorter than required to perform the above test, its value is implicitly extended with ones. Consequently, a view subtree family having a family mask of zero length always corresponds to a single view subtree.

2.5. Access Policy

The View-based Access Control Model determines the access rights of a group, representing zero or more securityNames which have the same access rights. For a particular context, identified by contextName, to which a group, identified by groupName, has access using a particular securityModel and securityLevel, that group's access rights are given by a read-view, a write-view and a notify-view.

The read-view represents the set of object instances authorized for the group when reading objects. Reading objects occurs when processing a retrieval operation (when handling Read Class PDUs).

The write-view represents the set of object instances authorized for the group when writing objects. Writing objects occurs when processing a write operation (when handling Write Class PDUs).

The notify-view represents the set of object instances authorized for the group when sending objects in a notification, such as when sending a notification (when sending Notification Class PDUs).

3. Elements of Procedure

This section describes the procedures followed by an Access Control module that implements the View-based Access Control Model when checking access rights as requested by an application (for example a Command Responder or a Notification Originator application). The abstract service primitive is:

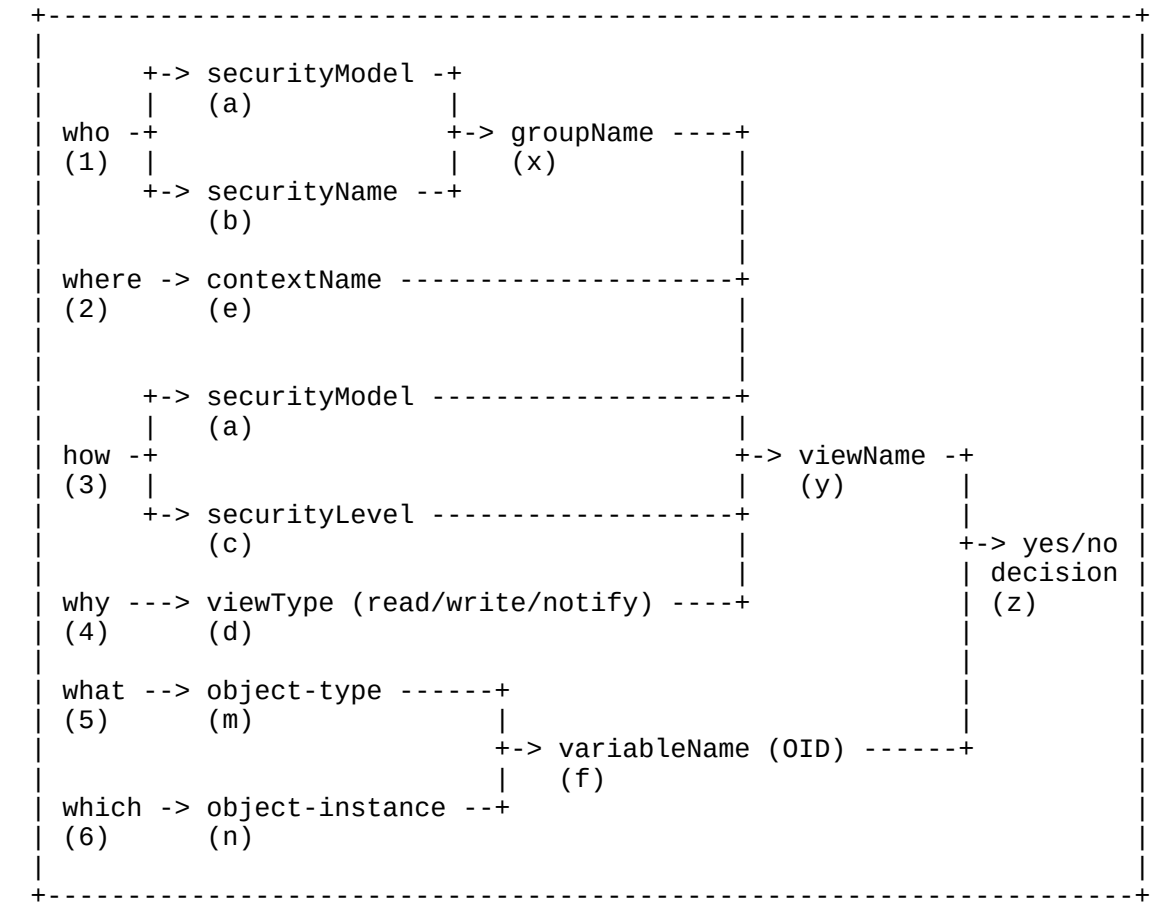
```
statusInformation =          -- success or errorIndication
    isAccessAllowed(
        securityModel        -- Security Model in use
        securityName         -- principal who wants access
        securityLevel        -- Level of Security
        viewType             -- read, write, or notify view
        contextName          -- context containing variableName
        variableName         -- OID for the managed object
        )
```

The abstract data elements are:

```
statusInformation - one of the following:
   accessAllowed  - a MIB view was found and access is granted.
   notInView      - a MIB view was found but access is denied.
                    The variableName is not in the configured
                    MIB view for the specified viewType (e.g., in
                    the relevant entry in the vacmAccessTable).
   noSuchView     - no MIB view found because no view has been
                    configured for specified viewType (e.g., in
                    the relevant entry in the vacmAccessTable).
   noSuchContext  - no MIB view found because of no entry in the
                    vacmContextTable for specified contextName.
   noGroupName    - no MIB view found because no entry has been
                    configured in the vacmSecurityToGroupTable
                    for the specified combination of
                    securityModel and securityName.
   noAccessEntry  - no MIB view found because no entry has been
                    configured in the vacmAccessTable for the
                    specified combination of contextName,
                    groupName (from vacmSecurityToGroupTable),
                    securityModel and securityLevel.
   otherError     - failure, an undefined error occurred.
securityModel - Security Model under which access is requested.
securityName  - the principal on whose behalf access is requested.
securityLevel - Level of Security under which access is requested.
viewType      - view to be checked (read, write or notify).
contextName   - context in which access is requested.
variableName  - object instance to which access is requested.
```

3.1. Overview of isAccessAllowed Process

The following picture shows how the decision for access control is made by the View-based Access Control Model.



How the decision for isAccessAllowed is made.

1) Inputs to the isAccessAllowed service are:

```
   (a)       securityModel    -- Security Model in use
   (b)       securityName     -- principal who wants to access
   (c)       securityLevel    -- Level of Security
   (d)       viewType         -- read, write, or notify view
   (e)       contextName      -- context containing variableName
   (f)       variableName     -- OID for the managed object
                              -- this is made up of:
                                 - object-type (m)
                                 - object-instance (n)
```

2) The partial "who" (1), represented by the securityModel (a) and the securityName (b), are used as the indices (a,b) into the vacmSecurityToGroupTable to find a single entry that produces a group, represented by groupName (x).

3) The "where" (2), represented by the contextName (e), the "who", represented by the groupName (x) from the previous step, and the "how" (3), represented by securityModel (a) and securityLevel (c), are used as indices (e,x,a,c) into the vacmAccessTable to find a single entry that contains three MIB views.

4) The "why" (4), represented by the viewType (d), is used to select the proper MIB view, represented by a viewName (y), from the vacmAccessEntry selected in the previous step. This viewName (y) is an index into the vacmViewTreeFamilyTable and selects the set of entries that define the variableNames which are included in or excluded from the MIB view identified by the viewName (y).

5) The "what" (5) type of management data and "which" (6) particular instance, represented by the variableName (f), is then checked to be in the MIB view or not, e.g., the yes/no decision (z).

3.2. Processing the isAccessAllowed Service Request

This section describes the procedure followed by an Access Control module that implements the View-based Access Control Model whenever it receives an isAccessAllowed request.

1) The vacmContextTable is consulted for information about the SNMP context identified by the contextName. If information about this SNMP context is absent from the table, then an errorIndication (noSuchContext) is returned to the calling module.

2) The vacmSecurityToGroupTable is consulted for mapping the securityModel and securityName to a groupName. If the information about this combination is absent from the table, then an errorIndication (noGroupName) is returned to the calling module.

3) The vacmAccessTable is consulted for information about the groupName, contextName, securityModel and securityLevel. If information about this combination is absent from the table, then an errorIndication (noAccessEntry) is returned to the calling module.

4) a) If the viewType is "read", then the read view is used for checking access rights.

   b) If the viewType is "write", then the write view is used for checking access rights.

   c) If the viewType is "notify", then the notify view is used for checking access rights.

   If the view to be used is the empty view (zero length viewName) then an errorIndication (noSuchView) is returned to the calling module.

5) a) If there is no view configured for the specified viewType, then an errorIndication (noSuchView) is returned to the calling module.

   b) If the specified variableName (object instance) is not in the MIB view (see DESCRIPTION clause for vacmViewTreeFamilyTable in section 4), then an errorIndication (notInView) is returned to the calling module.

      Otherwise,

   c) The specified variableName is in the MIB view. A statusInformation of success (accessAllowed) is returned to the calling module.

4. Definitions

```
SNMP-VIEW-BASED-ACM-MIB DEFINITIONS ::= BEGIN

IMPORTS
    MODULE-COMPLIANCE, OBJECT-GROUP       FROM SNMPv2-CONF
    MODULE-IDENTITY, OBJECT-TYPE,
    snmpModules                           FROM SNMPv2-SMI
    TestAndIncr,
    RowStatus, StorageType                FROM SNMPv2-TC
    SnmpAdminString,
    SnmpSecurityLevel,
    SnmpSecurityModel                     FROM SNMP-FRAMEWORK-MIB;

snmpVacmMIB       MODULE-IDENTITY
    LAST-UPDATED "200210160000Z"          -- 16 Oct 2002, midnight
    ORGANIZATION "SNMPv3 Working Group"
    CONTACT-INFO "WG-email:   snmpv3@lists.tislabs.com
                  Subscribe:  majordomo@lists.tislabs.com
                              In message body:  subscribe snmpv3

                  Co-Chair:   Russ Mundy
                              Network Associates Laboratories
                  postal:     15204 Omega Drive, Suite 300
                              Rockville, MD 20850-4601
                              USA
                  email:      mundy@tislabs.com
                  phone:      +1 301-947-7107

                  Co-Chair:   David Harrington
                              Enterasys Networks
                  Postal:     35 Industrial Way
                              P. O. Box 5004
                              Rochester, New Hampshire 03866-5005
                              USA
                  EMail:      dbh@enterasys.com
                  Phone:      +1 603-337-2614

                  Co-editor:  Bert Wijnen
                              Lucent Technologies
                  postal:     Schagen 33
                              3461 GL Linschoten
                              Netherlands
                  email:      bwijnen@lucent.com
                  phone:      +31-348-480-685

                  Co-editor:  Randy Presuhn
                              BMC Software, Inc.
```

```
                    postal:     2141 North First Street
                                San Jose, CA 95131
                                USA
                    email:      randy_presuhn@bmc.com
                    phone:      +1 408-546-1006

                    Co-editor:  Keith McCloghrie
                                Cisco Systems, Inc.
                    postal:     170 West Tasman Drive
                                San Jose, CA  95134-1706
                                USA
                    email:      kzm@cisco.com
                    phone:      +1-408-526-5260
                   "
    DESCRIPTION  "The management information definitions for the
                  View-based Access Control Model for SNMP.

                  Copyright (C) The Internet Society (2002). This
                  version of this MIB module is part of RFC 3415;
                  see the RFC itself for full legal notices.
                 "
--  Revision history

    REVISION     "200210160000Z"          -- 16 Oct 2002, midnight
    DESCRIPTION  "Clarifications, published as RFC3415"

    REVISION     "199901200000Z"          -- 20 Jan 1999, midnight
    DESCRIPTION  "Clarifications, published as RFC2575"

    REVISION     "199711200000Z"          -- 20 Nov 1997, midnight
    DESCRIPTION  "Initial version, published as RFC2275"

    ::= { snmpModules 16 }

-- Administrative assignments ****************************************

vacmMIBObjects      OBJECT IDENTIFIER ::= { snmpVacmMIB 1 }
vacmMIBConformance  OBJECT IDENTIFIER ::= { snmpVacmMIB 2 }

-- Information about Local Contexts **********************************

vacmContextTable OBJECT-TYPE
    SYNTAX       SEQUENCE OF VacmContextEntry
    MAX-ACCESS   not-accessible
    STATUS       current
    DESCRIPTION "The table of locally available contexts.

                 This table provides information to SNMP Command
```

```
                Generator applications so that they can properly
                configure the vacmAccessTable to control access to
                all contexts at the SNMP entity.

                This table may change dynamically if the SNMP entity
                allows that contexts are added/deleted dynamically
                (for instance when its configuration changes).  Such
                changes would happen only if the management
                instrumentation at that SNMP entity recognizes more
                (or fewer) contexts.

                The presence of entries in this table and of entries
                in the vacmAccessTable are independent.  That is, a
                context identified by an entry in this table is not
                necessarily referenced by any entries in the
                vacmAccessTable; and the context(s) referenced by an
                entry in the vacmAccessTable does not necessarily
                currently exist and thus need not be identified by an
                entry in this table.

                This table must be made accessible via the default
                context so that Command Responder applications have
                a standard way of retrieving the information.

                This table is read-only.  It cannot be configured via
                SNMP.
               "
    ::= { vacmMIBObjects 1 }

vacmContextEntry OBJECT-TYPE
    SYNTAX       VacmContextEntry
    MAX-ACCESS   not-accessible
    STATUS       current
    DESCRIPTION "Information about a particular context."
    INDEX       {
                  vacmContextName
                }
    ::= { vacmContextTable 1 }

VacmContextEntry ::= SEQUENCE
    {
        vacmContextName SnmpAdminString
    }

vacmContextName  OBJECT-TYPE
    SYNTAX       SnmpAdminString (SIZE(0..32))
    MAX-ACCESS   read-only
    STATUS       current
```

```
    DESCRIPTION "A human readable name identifying a particular
                 context at a particular SNMP entity.

                 The empty contextName (zero length) represents the
                 default context.
                "
    ::= { vacmContextEntry 1 }

-- Information about Groups ****************************************

vacmSecurityToGroupTable OBJECT-TYPE
    SYNTAX       SEQUENCE OF VacmSecurityToGroupEntry
    MAX-ACCESS   not-accessible
    STATUS       current
    DESCRIPTION "This table maps a combination of securityModel and
                 securityName into a groupName which is used to define
                 an access control policy for a group of principals.
                "
    ::= { vacmMIBObjects 2 }

vacmSecurityToGroupEntry OBJECT-TYPE
    SYNTAX       VacmSecurityToGroupEntry
    MAX-ACCESS   not-accessible
    STATUS       current
    DESCRIPTION "An entry in this table maps the combination of a
                 securityModel and securityName into a groupName.
                "
    INDEX       {
                  vacmSecurityModel,
                  vacmSecurityName
                }
    ::= { vacmSecurityToGroupTable 1 }

VacmSecurityToGroupEntry ::= SEQUENCE
    {
        vacmSecurityModel               SnmpSecurityModel,
        vacmSecurityName                SnmpAdminString,
        vacmGroupName                   SnmpAdminString,
        vacmSecurityToGroupStorageType  StorageType,
        vacmSecurityToGroupStatus       RowStatus
    }

vacmSecurityModel OBJECT-TYPE
    SYNTAX       SnmpSecurityModel(1..2147483647)
    MAX-ACCESS   not-accessible
    STATUS       current
    DESCRIPTION "The Security Model, by which the vacmSecurityName
                 referenced by this entry is provided.
```

```
                Note, this object may not take the 'any' (0) value.
               "
    ::= { vacmSecurityToGroupEntry 1 }

vacmSecurityName OBJECT-TYPE
    SYNTAX       SnmpAdminString (SIZE(1..32))
    MAX-ACCESS   not-accessible
    STATUS       current
    DESCRIPTION "The securityName for the principal, represented in a
                 Security Model independent format, which is mapped by
                 this entry to a groupName.
                "
    ::= { vacmSecurityToGroupEntry 2 }

vacmGroupName    OBJECT-TYPE
    SYNTAX       SnmpAdminString (SIZE(1..32))
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION "The name of the group to which this entry (e.g., the
                 combination of securityModel and securityName)
                 belongs.

                 This groupName is used as index into the
                 vacmAccessTable to select an access control policy.
                 However, a value in this table does not imply that an
                 instance with the value exists in table vacmAccesTable.
                "
    ::= { vacmSecurityToGroupEntry 3 }

vacmSecurityToGroupStorageType OBJECT-TYPE
    SYNTAX       StorageType
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION "The storage type for this conceptual row.
                 Conceptual rows having the value 'permanent' need not
                 allow write-access to any columnar objects in the row.
                "
    DEFVAL      { nonVolatile }
    ::= { vacmSecurityToGroupEntry 4 }

vacmSecurityToGroupStatus OBJECT-TYPE
    SYNTAX       RowStatus
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION "The status of this conceptual row.

                 Until instances of all corresponding columns are
                 appropriately configured, the value of the
```

```
                corresponding instance of the vacmSecurityToGroupStatus
                column is 'notReady'.

                In particular, a newly created row cannot be made
                active until a value has been set for vacmGroupName.

                The  RowStatus TC [RFC2579] requires that this
                DESCRIPTION clause states under which circumstances
                other objects in this row can be modified:

                The value of this object has no effect on whether
                other objects in this conceptual row can be modified.
              "
    ::= { vacmSecurityToGroupEntry 5 }

-- Information about Access Rights ************************************

vacmAccessTable  OBJECT-TYPE
    SYNTAX       SEQUENCE OF VacmAccessEntry
    MAX-ACCESS   not-accessible
    STATUS       current
    DESCRIPTION "The table of access rights for groups.

                Each entry is indexed by a groupName, a contextPrefix,
                a securityModel and a securityLevel.  To determine
                whether access is allowed, one entry from this table
                needs to be selected and the proper viewName from that
                entry must be used for access control checking.

                To select the proper entry, follow these steps:

                1) the set of possible matches is formed by the
                   intersection of the following sets of entries:

                     the set of entries with identical vacmGroupName
                     the union of these two sets:
                      - the set with identical vacmAccessContextPrefix
                      - the set of entries with vacmAccessContextMatch
                        value of 'prefix' and matching
                        vacmAccessContextPrefix
                     intersected with the union of these two sets:
                      - the set of entries with identical
                        vacmSecurityModel
                      - the set of entries with vacmSecurityModel
                        value of 'any'
                     intersected with the set of entries with
                     vacmAccessSecurityLevel value less than or equal
                     to the requested securityLevel
```

```
                 2) if this set has only one member, we're done
                    otherwise, it comes down to deciding how to weight
                    the preferences between ContextPrefixes,
                    SecurityModels, and SecurityLevels as follows:
                    a) if the subset of entries with securityModel
                       matching the securityModel in the message is
                       not empty, then discard the rest.
                    b) if the subset of entries with
                       vacmAccessContextPrefix matching the contextName
                       in the message is not empty,
                       then discard the rest
                    c) discard all entries with ContextPrefixes shorter
                       than the longest one remaining in the set
                    d) select the entry with the highest securityLevel

                 Please note that for securityLevel noAuthNoPriv, all
                 groups are really equivalent since the assumption that
                 the securityName has been authenticated does not hold.
                "
    ::= { vacmMIBObjects 4 }

vacmAccessEntry  OBJECT-TYPE
    SYNTAX       VacmAccessEntry
    MAX-ACCESS   not-accessible
    STATUS       current
    DESCRIPTION "An access right configured in the Local Configuration
                 Datastore (LCD) authorizing access to an SNMP context.

                 Entries in this table can use an instance value for
                 object vacmGroupName even if no entry in table
                 vacmAccessSecurityToGroupTable has a corresponding
                 value for object vacmGroupName.
                "
    INDEX       { vacmGroupName,
                  vacmAccessContextPrefix,
                  vacmAccessSecurityModel,
                  vacmAccessSecurityLevel
                }
    ::= { vacmAccessTable 1 }

VacmAccessEntry ::= SEQUENCE
    {
        vacmAccessContextPrefix    SnmpAdminString,
        vacmAccessSecurityModel    SnmpSecurityModel,
        vacmAccessSecurityLevel    SnmpSecurityLevel,
        vacmAccessContextMatch     INTEGER,
        vacmAccessReadViewName     SnmpAdminString,
        vacmAccessWriteViewName    SnmpAdminString,
```

```
        vacmAccessNotifyViewName   SnmpAdminString,
        vacmAccessStorageType      StorageType,
        vacmAccessStatus           RowStatus
    }

vacmAccessContextPrefix OBJECT-TYPE
    SYNTAX       SnmpAdminString (SIZE(0..32))
    MAX-ACCESS   not-accessible
    STATUS       current
    DESCRIPTION "In order to gain the access rights allowed by this
                 conceptual row, a contextName must match exactly
                 (if the value of vacmAccessContextMatch is 'exact')
                 or partially (if the value of vacmAccessContextMatch
                 is 'prefix') to the value of the instance of this
                 object.
                "
    ::= { vacmAccessEntry 1 }

vacmAccessSecurityModel OBJECT-TYPE
    SYNTAX       SnmpSecurityModel
    MAX-ACCESS   not-accessible
    STATUS       current
    DESCRIPTION "In order to gain the access rights allowed by this
                 conceptual row, this securityModel must be in use.
                "
    ::= { vacmAccessEntry 2 }

vacmAccessSecurityLevel OBJECT-TYPE
    SYNTAX       SnmpSecurityLevel
    MAX-ACCESS   not-accessible
    STATUS       current
    DESCRIPTION "The minimum level of security required in order to
                 gain the access rights allowed by this conceptual
                 row.  A securityLevel of noAuthNoPriv is less than
                 authNoPriv which in turn is less than authPriv.

                 If multiple entries are equally indexed except for
                 this vacmAccessSecurityLevel index, then the entry
                 which has the highest value for
                 vacmAccessSecurityLevel is selected.
                "
    ::= { vacmAccessEntry 3 }

vacmAccessContextMatch OBJECT-TYPE
    SYNTAX       INTEGER
                 { exact (1), -- exact match of prefix and contextName
                   prefix (2) -- Only match to the prefix
                 }
```

```
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION "If the value of this object is exact(1), then all
                 rows where the contextName exactly matches
                 vacmAccessContextPrefix are selected.

                 If the value of this object is prefix(2), then all
                 rows where the contextName whose starting octets
                 exactly match vacmAccessContextPrefix are selected.
                 This allows for a simple form of wildcarding.
                "
    DEFVAL      { exact }
    ::= { vacmAccessEntry 4 }

vacmAccessReadViewName OBJECT-TYPE
    SYNTAX       SnmpAdminString (SIZE(0..32))
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION "The value of an instance of this object identifies
                 the MIB view of the SNMP context to which this
                 conceptual row authorizes read access.

                 The identified MIB view is that one for which the
                 vacmViewTreeFamilyViewName has the same value as the
                 instance of this object; if the value is the empty
                 string or if there is no active MIB view having this
                 value of vacmViewTreeFamilyViewName, then no access
                 is granted.
                "
    DEFVAL      { ''H }   -- the empty string
    ::= { vacmAccessEntry 5 }

vacmAccessWriteViewName OBJECT-TYPE
    SYNTAX       SnmpAdminString (SIZE(0..32))
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION "The value of an instance of this object identifies
                 the MIB view of the SNMP context to which this
                 conceptual row authorizes write access.

                 The identified MIB view is that one for which the
                 vacmViewTreeFamilyViewName has the same value as the
                 instance of this object; if the value is the empty
                 string or if there is no active MIB view having this
                 value of vacmViewTreeFamilyViewName, then no access
                 is granted.
                "
    DEFVAL      { ''H }   -- the empty string
```

```
    ::= { vacmAccessEntry 6 }

vacmAccessNotifyViewName OBJECT-TYPE
    SYNTAX       SnmpAdminString (SIZE(0..32))
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION "The value of an instance of this object identifies
                 the MIB view of the SNMP context to which this
                 conceptual row authorizes access for notifications.

                 The identified MIB view is that one for which the
                 vacmViewTreeFamilyViewName has the same value as the
                 instance of this object; if the value is the empty
                 string or if there is no active MIB view having this
                 value of vacmViewTreeFamilyViewName, then no access
                 is granted.
                "
    DEFVAL      { ''H }   -- the empty string
    ::= { vacmAccessEntry 7 }

vacmAccessStorageType OBJECT-TYPE
    SYNTAX       StorageType
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION "The storage type for this conceptual row.

                 Conceptual rows having the value 'permanent' need not
                 allow write-access to any columnar objects in the row.
                "
    DEFVAL      { nonVolatile }
    ::= { vacmAccessEntry 8 }

vacmAccessStatus OBJECT-TYPE
    SYNTAX       RowStatus
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION "The status of this conceptual row.

                 The  RowStatus TC [RFC2579] requires that this
                 DESCRIPTION clause states under which circumstances
                 other objects in this row can be modified:

                 The value of this object has no effect on whether
                 other objects in this conceptual row can be modified.
                "
    ::= { vacmAccessEntry 9 }

-- Information about MIB views ************************************
```

```
-- Support for instance-level granularity is optional.
--
-- In some implementations, instance-level access control
-- granularity may come at a high performance cost.  Managers
-- should avoid requesting such configurations unnecessarily.

vacmMIBViews     OBJECT IDENTIFIER ::= { vacmMIBObjects 5 }

vacmViewSpinLock OBJECT-TYPE
    SYNTAX       TestAndIncr
    MAX-ACCESS   read-write
    STATUS       current
    DESCRIPTION "An advisory lock used to allow cooperating SNMP
                 Command Generator applications to coordinate their
                 use of the Set operation in creating or modifying
                 views.

                 When creating a new view or altering an existing
                 view, it is important to understand the potential
                 interactions with other uses of the view.  The
                 vacmViewSpinLock should be retrieved.  The name of
                 the view to be created should be determined to be
                 unique by the SNMP Command Generator application by
                 consulting the vacmViewTreeFamilyTable.  Finally,
                 the named view may be created (Set), including the
                 advisory lock.
                 If another SNMP Command Generator application has
                 altered the views in the meantime, then the spin
                 lock's value will have changed, and so this creation
                 will fail because it will specify the wrong value for
                 the spin lock.

                 Since this is an advisory lock, the use of this lock
                 is not enforced.
                "
    ::= { vacmMIBViews 1 }

vacmViewTreeFamilyTable OBJECT-TYPE
    SYNTAX       SEQUENCE OF VacmViewTreeFamilyEntry
    MAX-ACCESS   not-accessible
    STATUS       current
    DESCRIPTION "Locally held information about families of subtrees
                 within MIB views.

                 Each MIB view is defined by two sets of view subtrees:
                   - the included view subtrees, and
                   - the excluded view subtrees.
                 Every such view subtree, both the included and the
```

excluded ones, is defined in this table.

To determine if a particular object instance is in a particular MIB view, compare the object instance's OBJECT IDENTIFIER with each of the MIB view's active entries in this table. If none match, then the object instance is not in the MIB view. If one or more match, then the object instance is included in, or excluded from, the MIB view according to the value of vacmViewTreeFamilyType in the entry whose value of vacmViewTreeFamilySubtree has the most sub-identifiers. If multiple entries match and have the same number of sub-identifiers (when wildcarding is specified with the value of vacmViewTreeFamilyMask), then the lexicographically greatest instance of vacmViewTreeFamilyType determines the inclusion or exclusion.

An object instance's OBJECT IDENTIFIER X matches an active entry in this table when the number of sub-identifiers in X is at least as many as in the value of vacmViewTreeFamilySubtree for the entry, and each sub-identifier in the value of vacmViewTreeFamilySubtree matches its corresponding sub-identifier in X. Two sub-identifiers match either if the corresponding bit of the value of vacmViewTreeFamilyMask for the entry is zero (the 'wild card' value), or if they are equal.

A 'family' of subtrees is the set of subtrees defined by a particular combination of values of vacmViewTreeFamilySubtree and vacmViewTreeFamilyMask.

In the case where no 'wild card' is defined in the vacmViewTreeFamilyMask, the family of subtrees reduces to a single subtree.

When creating or changing MIB views, an SNMP Command Generator application should utilize the vacmViewSpinLock to try to avoid collisions. See DESCRIPTION clause of vacmViewSpinLock.

When creating MIB views, it is strongly advised that first the 'excluded' vacmViewTreeFamilyEntries are created and then the 'included' entries.

When deleting MIB views, it is strongly advised that first the 'included' vacmViewTreeFamilyEntries are

```
                deleted and then the 'excluded' entries.

                If a create for an entry for instance-level access
                control is received and the implementation does not
                support instance-level granularity, then an
                inconsistentName error must be returned.
               "
    ::= { vacmMIBViews 2 }

vacmViewTreeFamilyEntry OBJECT-TYPE
    SYNTAX       VacmViewTreeFamilyEntry
    MAX-ACCESS   not-accessible
    STATUS       current
    DESCRIPTION "Information on a particular family of view subtrees
                 included in or excluded from a particular SNMP
                 context's MIB view.

                 Implementations must not restrict the number of
                 families of view subtrees for a given MIB view,
                 except as dictated by resource constraints on the
                 overall number of entries in the
                 vacmViewTreeFamilyTable.

                 If no conceptual rows exist in this table for a given
                 MIB view (viewName), that view may be thought of as
                 consisting of the empty set of view subtrees.
                "
    INDEX       { vacmViewTreeFamilyViewName,
                  vacmViewTreeFamilySubtree
                }
    ::= { vacmViewTreeFamilyTable 1 }

VacmViewTreeFamilyEntry ::= SEQUENCE
    {
        vacmViewTreeFamilyViewName     SnmpAdminString,
        vacmViewTreeFamilySubtree      OBJECT IDENTIFIER,
        vacmViewTreeFamilyMask         OCTET STRING,
        vacmViewTreeFamilyType         INTEGER,
        vacmViewTreeFamilyStorageType  StorageType,
        vacmViewTreeFamilyStatus       RowStatus
    }

vacmViewTreeFamilyViewName OBJECT-TYPE
    SYNTAX       SnmpAdminString (SIZE(1..32))
    MAX-ACCESS   not-accessible
    STATUS       current
    DESCRIPTION "The human readable name for a family of view subtrees.
                "
```

```
    ::= { vacmViewTreeFamilyEntry 1 }

vacmViewTreeFamilySubtree OBJECT-TYPE
    SYNTAX       OBJECT IDENTIFIER
    MAX-ACCESS   not-accessible
    STATUS       current
    DESCRIPTION "The MIB subtree which when combined with the
                 corresponding instance of vacmViewTreeFamilyMask
                 defines a family of view subtrees.
                "
    ::= { vacmViewTreeFamilyEntry 2 }

vacmViewTreeFamilyMask OBJECT-TYPE
    SYNTAX       OCTET STRING (SIZE (0..16))
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION "The bit mask which, in combination with the
                 corresponding instance of vacmViewTreeFamilySubtree,
                 defines a family of view subtrees.

                 Each bit of this bit mask corresponds to a
                 sub-identifier of vacmViewTreeFamilySubtree, with the
                 most significant bit of the i-th octet of this octet
                 string value (extended if necessary, see below)
                 corresponding to the (8*i - 7)-th sub-identifier, and
                 the least significant bit of the i-th octet of this
                 octet string corresponding to the (8*i)-th
                 sub-identifier, where i is in the range 1 through 16.

                 Each bit of this bit mask specifies whether or not
                 the corresponding sub-identifiers must match when
                 determining if an OBJECT IDENTIFIER is in this
                 family of view subtrees; a '1' indicates that an
                 exact match must occur; a '0' indicates 'wild card',
                 i.e., any sub-identifier value matches.

                 Thus, the OBJECT IDENTIFIER X of an object instance
                 is contained in a family of view subtrees if, for
                 each sub-identifier of the value of
                 vacmViewTreeFamilySubtree, either:

                   the i-th bit of vacmViewTreeFamilyMask is 0, or

                   the i-th sub-identifier of X is equal to the i-th
                   sub-identifier of the value of
                   vacmViewTreeFamilySubtree.

                 If the value of this bit mask is M bits long and
```

```
                there are more than M sub-identifiers in the
                corresponding instance of vacmViewTreeFamilySubtree,
                then the bit mask is extended with 1's to be the
                required length.

                Note that when the value of this object is the
                zero-length string, this extension rule results in
                a mask of all-1's being used (i.e., no 'wild card'),
                and the family of view subtrees is the one view
                subtree uniquely identified by the corresponding
                instance of vacmViewTreeFamilySubtree.

                Note that masks of length greater than zero length
                do not need to be supported.  In this case this
                object is made read-only.
               "
    DEFVAL      { ''H }
    ::= { vacmViewTreeFamilyEntry 3 }

vacmViewTreeFamilyType OBJECT-TYPE
    SYNTAX       INTEGER  { included(1), excluded(2) }
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION "Indicates whether the corresponding instances of
                 vacmViewTreeFamilySubtree and vacmViewTreeFamilyMask
                 define a family of view subtrees which is included in
                 or excluded from the MIB view.
                "
    DEFVAL      { included }
    ::= { vacmViewTreeFamilyEntry 4 }

vacmViewTreeFamilyStorageType OBJECT-TYPE
    SYNTAX       StorageType
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION "The storage type for this conceptual row.

                 Conceptual rows having the value 'permanent' need not
                 allow write-access to any columnar objects in the row.
                "
    DEFVAL      { nonVolatile }
    ::= { vacmViewTreeFamilyEntry 5 }

vacmViewTreeFamilyStatus OBJECT-TYPE
    SYNTAX       RowStatus
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION "The status of this conceptual row.
```

```
                The  RowStatus TC [RFC2579] requires that this
                DESCRIPTION clause states under which circumstances
                other objects in this row can be modified:

                The value of this object has no effect on whether
                other objects in this conceptual row can be modified.
               "
    ::= { vacmViewTreeFamilyEntry 6 }

-- Conformance information ********************************************

vacmMIBCompliances  OBJECT IDENTIFIER ::= { vacmMIBConformance 1 }
vacmMIBGroups       OBJECT IDENTIFIER ::= { vacmMIBConformance 2 }

-- Compliance statements **********************************************

vacmMIBCompliance MODULE-COMPLIANCE
    STATUS       current
    DESCRIPTION "The compliance statement for SNMP engines which
                 implement the SNMP View-based Access Control Model
                 configuration MIB.
                "
    MODULE -- this module
        MANDATORY-GROUPS { vacmBasicGroup }

        OBJECT        vacmAccessContextMatch
        MIN-ACCESS    read-only
        DESCRIPTION  "Write access is not required."

        OBJECT        vacmAccessReadViewName
        MIN-ACCESS    read-only
        DESCRIPTION  "Write access is not required."

        OBJECT        vacmAccessWriteViewName
        MIN-ACCESS    read-only
        DESCRIPTION  "Write access is not required."

        OBJECT        vacmAccessNotifyViewName
        MIN-ACCESS    read-only
        DESCRIPTION  "Write access is not required."

        OBJECT        vacmAccessStorageType
        MIN-ACCESS    read-only
        DESCRIPTION  "Write access is not required."

        OBJECT        vacmAccessStatus
        MIN-ACCESS    read-only
        DESCRIPTION  "Create/delete/modify access to the
```

```
                     vacmAccessTable is not required.
                    "

        OBJECT        vacmViewTreeFamilyMask
        WRITE-SYNTAX  OCTET STRING (SIZE (0))
        MIN-ACCESS    read-only
        DESCRIPTION  "Support for configuration via SNMP of subtree
                      families using wild-cards is not required.
                    "

        OBJECT        vacmViewTreeFamilyType
        MIN-ACCESS    read-only
        DESCRIPTION  "Write access is not required."

        OBJECT        vacmViewTreeFamilyStorageType
        MIN-ACCESS    read-only
        DESCRIPTION  "Write access is not required."

        OBJECT        vacmViewTreeFamilyStatus
        MIN-ACCESS    read-only
        DESCRIPTION  "Create/delete/modify access to the
                      vacmViewTreeFamilyTable is not required.
                    "
    ::= { vacmMIBCompliances 1 }

-- Units of conformance ********************************************

vacmBasicGroup OBJECT-GROUP
    OBJECTS {
              vacmContextName,
              vacmGroupName,
              vacmSecurityToGroupStorageType,
              vacmSecurityToGroupStatus,
              vacmAccessContextMatch,
              vacmAccessReadViewName,
              vacmAccessWriteViewName,
              vacmAccessNotifyViewName,
              vacmAccessStorageType,
              vacmAccessStatus,
              vacmViewSpinLock,
              vacmViewTreeFamilyMask,
              vacmViewTreeFamilyType,
              vacmViewTreeFamilyStorageType,
              vacmViewTreeFamilyStatus
            }
    STATUS       current
    DESCRIPTION "A collection of objects providing for remote
                 configuration of an SNMP engine which implements
```

```
                the SNMP View-based Access Control Model.
               "
    ::= { vacmMIBGroups 1 }

END
```

5. Intellectual Property

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-related documentation can be found in BCP-11. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to 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.

6. Acknowledgements

This document is the result of the efforts of the SNMPv3 Working Group. Some special thanks are in order to the following SNMPv3 WG members:

Harald Tveit Alvestrand (Maxware)
Dave Battle (SNMP Research, Inc.)
Alan Beard (Disney Worldwide Services)
Paul Berrevoets (SWI Systemware/Halcyon Inc.)
Martin Bjorklund (Ericsson)
Uri Blumenthal (IBM T.J. Watson Research Center)
Jeff Case (SNMP Research, Inc.)
John Curran (BBN)
Mike Daniele (Compaq Computer Corporation)
T. Max Devlin (Eltrax Systems)
John Flick (Hewlett Packard)
Rob Frye (MCI)
Wes Hardaker (U.C.Davis, Information Technology - D.C.A.S.)
David Harrington (Cabletron Systems Inc.)

Lauren Heintz (BMC Software, Inc.)
N.C. Hien (IBM T.J. Watson Research Center)
Michael Kirkham (InterWorking Labs, Inc.)
Dave Levi (SNMP Research, Inc.)
Louis A Mamakos (UUNET Technologies Inc.)
Joe Marzot (Nortel Networks)
Paul Meyer (Secure Computing Corporation)
Keith McCloghrie (Cisco Systems)
Bob Moore (IBM)
Russ Mundy (TIS Labs at Network Associates)
Bob Natale (ACE*COMM Corporation)
Mike O'Dell (UUNET Technologies Inc.)
Dave Perkins (DeskTalk)
Peter Polkinghorne (Brunel University)
Randy Presuhn (BMC Software, Inc.)
David Reeder (TIS Labs at Network Associates)
David Reid (SNMP Research, Inc.)
Aleksey Romanov (Quality Quorum)
Shawn Routhier (Epilogue)
Juergen Schoenwaelder (TU Braunschweig)
Bob Stewart (Cisco Systems)
Mike Thatcher (Independent Consultant)
Bert Wijnen (IBM T.J. Watson Research Center)

The document is based on recommendations of the IETF Security and Administrative Framework Evolution for SNMP Advisory Team. Members of that Advisory Team were:

David Harrington (Cabletron Systems Inc.)
Jeff Johnson (Cisco Systems)
David Levi (SNMP Research Inc.)
John Linn (Openvision)
Russ Mundy (Trusted Information Systems) chair
Shawn Routhier (Epilogue)
Glenn Waters (Nortel)
Bert Wijnen (IBM T. J. Watson Research Center)

As recommended by the Advisory Team and the SNMPv3 Working Group Charter, the design incorporates as much as practical from previous RFCs and drafts. As a result, special thanks are due to the authors of previous designs known as SNMPv2u and SNMPv2*:

Jeff Case (SNMP Research, Inc.)
David Harrington (Cabletron Systems Inc.)
David Levi (SNMP Research, Inc.)
Keith McCloghrie (Cisco Systems)
Brian O'Keefe (Hewlett Packard)
Marshall T. Rose (Dover Beach Consulting)

Jon Saperia (BGS Systems Inc.)
Steve Waldbusser (International Network Services)
Glenn W. Waters (Bell-Northern Research Ltd.)

7. Security Considerations

7.1. Recommended Practices

This document is meant for use in the SNMP architecture. The View-based Access Control Model described in this document checks access rights to management information based on:

- contextName, representing a set of management information at the managed system where the Access Control module is running.

- groupName, representing a set of zero or more securityNames. The combination of a securityModel and a securityName is mapped into a group in the View-based Access Control Model.

- securityModel under which access is requested.

- securityLevel under which access is requested.

- operation performed on the management information.

- MIB views for read, write or notify access.

When the User-based Access Control module is called for checking access rights, it is assumed that the calling module has ensured the authentication and privacy aspects as specified by the securityLevel that is being passed.

When creating entries in or deleting entries from the vacmViewTreeFamilyTable it is important to do such in the sequence as recommended in the DESCRIPTION clause of the vacmViewTreeFamilyTable definition. Otherwise unwanted access may be granted while changing the entries in the table.

7.2. Defining Groups

The groupNames are used to give access to a group of zero or more securityNames. Within the View-Based Access Control Model, a groupName is considered to exist if that groupName is listed in the vacmSecurityToGroupTable.

By mapping the combination of a securityModel and securityName into a groupName, an SNMP Command Generator application can add/delete securityNames to/from a group, if proper access is allowed.

Further it is important to realize that the grouping of <securityModel, securityName> tuples in the vacmSecurityToGroupTable does not take securityLevel into account. It is therefore important that the security administrator uses the securityLevel index in the vacmAccessTable to separate noAuthNoPriv from authPriv and/or authNoPriv access.

7.3. Conformance

For an implementation of the View-based Access Control Model to be conformant, it MUST implement the SNMP-VIEW-BASED-ACM-MIB according to the vacmMIBCompliance. It also SHOULD implement the initial configuration, described in appendix A.

7.4. Access to the SNMP-VIEW-BASED-ACM-MIB

The objects in this MIB control the access to all MIB data that is accessible via the SNMP engine and they may be considered sensitive in many environments. It is important to closely control (both read and write) access to these to these MIB objects by using appropriately configured Access Control models (for example the View-based Access Control Model as specified in this document).

8. References

8.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2578] McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M. and S. Waldbusser, "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, April 1999.

[RFC2579] McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M. and S. Waldbusser, "Textual Conventions for SMIv2", STD 58, RFC 2579, April 1999.

[RFC2580] McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M. and S. Waldbusser, "Conformance Statements for SMIv2", STD 58, RFC 2580, April 1999.

[RFC3411] Harrington, D., Presuhn, R. and B. Wijnen, "An Architecture for describing Simple Network Management Protocol (SNMP) Management Frameworks", STD 62, RFC 3411, December 2002.

[SNMP3412] 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.

[RFC3414] 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.

8.2. Informative References

[ISO-ASN.1] Information processing systems - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1), International Organization for Standardization. International Standard 8824, (December, 1987).

Appendix A - Installation

A.1. Installation Parameters

During installation, an authoritative SNMP engine which supports this View-based Access Control Model SHOULD be configured with several initial parameters. These include for the View-based Access Control Model:

1) A security configuration

   The choice of security configuration determines if initial configuration is implemented and if so how. One of three possible choices is selected:

   - initial-minimum-security-configuration
   - initial-semi-security-configuration
   - initial-no-access-configuration

   In the case of a initial-no-access-configuration, there is no initial configuration, and so the following steps are irrelevant.

2) A default context

   One entry in the vacmContextTable with a contextName of "" (the empty string), representing the default context. Note that this table gets created automatically if a default context exists.

vacmContextName	""

3) An initial group

   One entry in the vacmSecurityToGroupTable to allow access to group "initial".

vacmSecurityModel	3 (USM)
vacmSecurityName	"initial"
vacmGroupName	"initial"
vacmSecurityToGroupStorageType	anyValidStorageType
vacmSecurityToGroupStatus	active

4) Initial access rights

Three entries in the vacmAccessTable as follows:

- read-notify access for securityModel USM, securityLevel "noAuthNoPriv" on behalf of securityNames that belong to the group "initial" to the <restricted> MIB view in the default context with contextName "".

- read-write-notify access for securityModel USM, securityLevel "authNoPriv" on behalf of securityNames that belong to the group "initial" to the <internet> MIB view in the default context with contextName "".

- if privacy is supported, read-write-notify access for securityModel USM, securityLevel "authPriv" on behalf of securityNames that belong to the group "initial" to the <internet> MIB view in the default context with contextName "".

That translates into the following entries in the vacmAccessTable.

- One entry to be used for unauthenticated access (noAuthNoPriv):

vacmGroupName	"initial"
vacmAccessContextPrefix	""
vacmAccessSecurityModel	3 (USM)
vacmAccessSecurityLevel	noAuthNoPriv
vacmAccessContextMatch	exact
vacmAccessReadViewName	"restricted"
vacmAccessWriteViewName	""
vacmAccessNotifyViewName	"restricted"
vacmAccessStorageType	anyValidStorageType
vacmAccessStatus	active

- One entry to be used for authenticated access (authNoPriv) with optional privacy (authPriv):

vacmGroupName	"initial"
vacmAccessContextPrefix	""
vacmAccessSecurityModel	3 (USM)
vacmAccessSecurityLevel	authNoPriv
vacmAccessContextMatch	exact
vacmAccessReadViewName	"internet"
vacmAccessWriteViewName	"internet"
vacmAccessNotifyViewName	"internet"
vacmAccessStorageType	anyValidStorageType
vacmAccessStatus	active

5) Two MIB views, of which the second one depends on the security configuration.

- One view, the <internet> view, for authenticated access:

  - the <internet> MIB view is the following subtree:
    "internet" (subtree 1.3.6.1)

- A second view, the <restricted> view, for unauthenticated access. This view is configured according to the selected security configuration:

- For the initial-no-access-configuration there is no default initial configuration, so no MIB views are pre-scribed.

- For the initial-semi-secure-configuration:

  the <restricted> MIB view is the union of these subtrees:
  (a) "system" (subtree 1.3.6.1.2.1.1) [RFC3918]
  (b) "snmp" (subtree 1.3.6.1.2.1.11) [RFC3918]
  (c) "snmpEngine" (subtree 1.3.6.1.6.3.10.2.1) [RFC3411]
  (d) "snmpMPDStats" (subtree 1.3.6.1.6.3.11.2.1) [RFC3412]
  (e) "usmStats" (subtree 1.3.6.1.6.3.15.1.1) [RFC3414]

- For the initial-minimum-secure-configuration:

  the <restricted> MIB view is the following subtree.
  "internet" (subtree 1.3.6.1)

This translates into the following "internet" entry in the vacmViewTreeFamilyTable:

	minimum-secure	semi-secure
vacmViewTreeFamilyViewName	"internet"	"internet"
vacmViewTreeFamilySubtree	1.3.6.1	1.3.6.1
vacmViewTreeFamilyMask	""	""
vacmViewTreeFamilyType	1 (included)	1 (included)
vacmViewTreeFamilyStorageType	anyValidStorageType	anyValidStorageType
vacmViewTreeFamilyStatus	active	active

In addition it translates into the following "restricted" entries in the vacmViewTreeFamilyTable:

	minimum-secure	semi-secure
vacmViewTreeFamilyViewName	"restricted"	"restricted"
vacmViewTreeFamilySubtree	1.3.6.1	1.3.6.1.2.1.1
vacmViewTreeFamilyMask	""	""
vacmViewTreeFamilyType	1 (included)	1 (included)
vacmViewTreeFamilyStorageType	anyValidStorageType	anyValidStorageType
vacmViewTreeFamilyStatus	active	active
vacmViewTreeFamilyViewName		"restricted"
vacmViewTreeFamilySubtree		1.3.6.1.2.1.11
vacmViewTreeFamilyMask		""
vacmViewTreeFamilyType		1 (included)
vacmViewTreeFamilyStorageType		anyValidStorageType
vacmViewTreeFamilyStatus		active
vacmViewTreeFamilyViewName		"restricted"
vacmViewTreeFamilySubtree		1.3.6.1.6.3.10.2.1
vacmViewTreeFamilyMask		""
vacmViewTreeFamilyType		1 (included)
vacmViewTreeFamilyStorageType		anyValidStorageType
vacmViewTreeFamilyStatus		active
vacmViewTreeFamilyViewName		"restricted"
vacmViewTreeFamilySubtree		1.3.6.1.6.3.11.2.1
vacmViewTreeFamilyMask		""
vacmViewTreeFamilyType		1 (included)
vacmViewTreeFamilyStorageType		anyValidStorageType
vacmViewTreeFamilyStatus		active
vacmViewTreeFamilyViewName		"restricted"
vacmViewTreeFamilySubtree		1.3.6.1.6.3.15.1.1
vacmViewTreeFamilyMask		""
vacmViewTreeFamilyType		1 (included)
vacmViewTreeFamilyStorageType		anyValidStorageType
vacmViewTreeFamilyStatus		active

B. Change Log

Changes made since RFC 2575:

- Removed reference from abstract as per RFC-Editor guidelines
- Updated references

Changes made since RFC 2275:

- Added text to vacmSecurityToGroupStatus DESCRIPTION clause to clarify under which conditions an entry in the vacmSecurityToGroupTable can be made active.
- Added REVISION clauses to MODULE-IDENTITY
- Clarified text in vacmAccessTable DESCRIPTION clause.
- Added a DEFVAL clause to vacmAccessContextMatch object.
- Added missing columns in Appendix A and re-arranged for clarity.
- Fixed oids in appendix A.
- Use the PDU Class terminology instead of RFC1905 PDU types.
- Added section 7.4 about access control to the MIB.
- Fixed references to new/revised documents
- Fix Editor contact information.
- fixed spelling errors
- removed one vacmAccesEntry from sample in appendix A.
- made some more clarifications.
- updated acknowledgement section.

Editors' Addresses

Bert Wijnen
Lucent Technologies
Schagen 33
3461 GL Linschoten
Netherlands

Phone: +31-348-480-685
EMail: bwijnen@lucent.com

Randy Presuhn
BMC Software, Inc.
2141 North First Street
San Jose, CA 95131
USA

Phone: +1 408-546-1006
EMail: randy_presuhn@bmc.com

Keith McCloghrie
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA

Phone: +1-408-526-5260
EMail: kzm@cisco.com

Full Copyright Statement

Acknowledgement

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

Network Working Group
Request for Comments: 3416
STD: 62
Obsoletes: 1905
Category: Standards Track

Editor of this version:
R. Presuhn
BMC Software, Inc.
Authors of previous version:
J. Case
SNMP Research, Inc.
K. McCloghrie
Cisco Systems, Inc.
M. Rose
Dover Beach Consulting, Inc.
S. Waldbusser
International Network Services
December 2002

Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)

Status of this Memo

Copyright Notice

Abstract

This document defines version 2 of the protocol operations for the Simple Network Management Protocol (SNMP). It defines the syntax and elements of procedure for sending, receiving, and processing SNMP PDUs. This document obsoletes RFC 1905.

Table of Contents

1. Introduction

The SNMP Management Framework at the time of this writing consists of five major components:

- An overall architecture, described in STD 62, RFC 3411 [RFC3411].

- 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 [RFC1155], STD 16, RFC 1212 [RFC1212] and RFC 1215 [RFC1215]. The second version, called SMIv2, is described in STD 58, RFC 2578 [RFC2578], STD 58, RFC 2579 [RFC2579] and STD 58, RFC 2580 [RFC2580].

- Message protocols for transferring management information. The first version of the SNMP message protocol is called SNMPv1 and described in STD 15, RFC 1157 [RFC1157]. A second version of the SNMP message protocol, which is not an Internet standards track protocol, is called SNMPv2c and described in RFC 1901 [RFC1901] and STD 62, RFC 3417 [RFC3417]. The third version of the message protocol is called SNMPv3 and described in STD 62, RFC 3417 [RFC3417], RFC 3412 [RFC3412] and RFC 3414 [RFC3414].

- Protocol operations for accessing management information. The first set of protocol operations and associated PDU formats is described in STD 15, RFC 1157 [RFC1157]. A second set of protocol operations and associated PDU formats is described in this document.

- A set of fundamental applications described in STD 62, RFC 3413 [RFC3413] and the view-based access control mechanism described in STD 62, RFC 3415 [RFC3415].

A more detailed introduction to the SNMP Management Framework at the time of this writing can be found in RFC 3410 [RFC3410].

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.

This document, Version 2 of the Protocol Operations for the Simple Network Management Protocol, defines the operations of the protocol with respect to the sending and receiving of PDUs to be carried by the message protocol.

2. Overview

SNMP entities supporting command generator or notification receiver applications (traditionally called "managers") communicate with SNMP entities supporting command responder or notification originator applications (traditionally called "agents"). The purpose of this protocol is the transport of management information and operations.

2.1. Management Information

The term "variable" refers to an instance of a non-aggregate object type defined according to the conventions set forth in the SMI [RFC2578] or the textual conventions based on the SMI [RFC2579]. The term "variable binding" normally refers to the pairing of the name of a variable and its associated value. However, if certain kinds of exceptional conditions occur during processing of a retrieval request, a variable binding will pair a name and an indication of that exception.

A variable-binding list is a simple list of variable bindings.

The name of a variable is an OBJECT IDENTIFIER which is the concatenation of the OBJECT IDENTIFIER of the corresponding object-type together with an OBJECT IDENTIFIER fragment identifying the instance. The OBJECT IDENTIFIER of the corresponding object-type is called the OBJECT IDENTIFIER prefix of the variable.

2.2. Retransmission of Requests

For all types of request in this protocol, the receiver is required under normal circumstances, to generate and transmit a response to the originator of the request. Whether or not a request should be retransmitted if no corresponding response is received in an appropriate time interval, is at the discretion of the application originating the request. This will normally depend on the urgency of the request. However, such an application needs to act responsibly in respect to the frequency and duration of re-transmissions. See BCP 41 [RFC2914] for discussion of relevant congestion control principles.

2.3. Message Sizes

The maximum size of an SNMP message is limited to the minimum of:

(1) the maximum message size which the destination SNMP entity can accept; and,

(2) the maximum message size which the source SNMP entity can generate.

The former may be known on a per-recipient basis; and in the absence of such knowledge, is indicated by transport domain used when sending the message. The latter is imposed by implementation-specific local constraints.

Each transport mapping for the SNMP indicates the minimum message size which a SNMP implementation must be able to produce or consume. Although implementations are encouraged to support larger values whenever possible, a conformant implementation must never generate messages larger than allowed by the receiving SNMP entity.

One of the aims of the GetBulkRequest-PDU, specified in this protocol, is to minimize the number of protocol exchanges required to retrieve a large amount of management information. As such, this PDU type allows an SNMP entity supporting command generator applications to request that the response be as large as possible given the constraints on message sizes. These constraints include the limits on the size of messages which the SNMP entity supporting command responder applications can generate, and the SNMP entity supporting command generator applications can receive.

However, it is possible that such maximum sized messages may be larger than the Path MTU of the path across the network traversed by the messages. In this situation, such messages are subject to fragmentation. Fragmentation is generally considered to be harmful [FRAG], since among other problems, it leads to a decrease in the reliability of the transfer of the messages. Thus, an SNMP entity which sends a GetBulkRequest-PDU must take care to set its parameters accordingly, so as to reduce the risk of fragmentation. In particular, under conditions of network stress, only small values should be used for max-repetitions.

2.4. Transport Mappings

It is important to note that the exchange of SNMP messages requires only an unreliable datagram service, with every message being entirely and independently contained in a single transport datagram. Specific transport mappings and encoding rules are specified elsewhere [RFC3417]. However, the preferred mapping is the use of the User Datagram Protocol [RFC768].

2.5. SMIv2 Data Type Mappings

The SMIv2 [RFC2578] defines 11 base types (INTEGER, OCTET STRING, OBJECT IDENTIFIER, Integer32, IpAddress, Counter32, Gauge32, Unsigned32, TimeTicks, Opaque, Counter64) and the BITS construct. The SMIv2 base types are mapped to the corresponding selection type in the SimpleSyntax and ApplicationSyntax choices of the ASN.1 SNMP protocol definition. Note that the INTEGER and Integer32 SMIv2 base types are mapped to the integer-value selection type of the SimpleSyntax choice. Similarly, the Gauge32 and Unsigned32 SMIv2 base types are mapped to the unsigned-integer-value selection type of the ApplicationSyntax choice.

The SMIv2 BITS construct is mapped to the string-value selection type of the SimpleSyntax choice. A BITS value is encoded as an OCTET STRING, in which all the named bits in (the definition of) the bitstring, commencing with the first bit and proceeding to the last bit, are placed in bits 8 (high order bit) to 1 (low order bit) of the first octet, followed by bits 8 to 1 of each subsequent octet in turn, followed by as many bits as are needed of the final subsequent octet, commencing with bit 8. Remaining bits, if any, of the final octet are set to zero on generation and ignored on receipt.

3. Definitions

The PDU syntax is defined using ASN.1 notation [ASN1].

```
SNMPv2-PDU DEFINITIONS ::= BEGIN

ObjectName ::= OBJECT IDENTIFIER

ObjectSyntax ::= CHOICE {
      simple           SimpleSyntax,
      application-wide ApplicationSyntax }

SimpleSyntax ::= CHOICE {
      integer-value   INTEGER (-2147483648..2147483647),
      string-value    OCTET STRING (SIZE (0..65535)),
      objectID-value  OBJECT IDENTIFIER }

ApplicationSyntax ::= CHOICE {
      ipAddress-value        IpAddress,
      counter-value          Counter32,
      timeticks-value        TimeTicks,
      arbitrary-value        Opaque,
      big-counter-value      Counter64,
      unsigned-integer-value Unsigned32 }
```

```
   IpAddress ::= [APPLICATION 0] IMPLICIT OCTET STRING (SIZE (4))

   Counter32 ::= [APPLICATION 1] IMPLICIT INTEGER (0..4294967295)

   Unsigned32 ::= [APPLICATION 2] IMPLICIT INTEGER (0..4294967295)

   Gauge32 ::= Unsigned32

   TimeTicks ::= [APPLICATION 3] IMPLICIT INTEGER (0..4294967295)

   Opaque ::= [APPLICATION 4] IMPLICIT OCTET STRING

   Counter64 ::= [APPLICATION 6]
                 IMPLICIT INTEGER (0..18446744073709551615)

   -- protocol data units

   PDUs ::= CHOICE {
        get-request      GetRequest-PDU,
        get-next-request GetNextRequest-PDU,
        get-bulk-request GetBulkRequest-PDU,
        response         Response-PDU,
        set-request      SetRequest-PDU,
        inform-request   InformRequest-PDU,
        snmpV2-trap      SNMPv2-Trap-PDU,
        report           Report-PDU }

   -- PDUs

   GetRequest-PDU ::= [0] IMPLICIT PDU

   GetNextRequest-PDU ::= [1] IMPLICIT PDU

   Response-PDU ::= [2] IMPLICIT PDU

   SetRequest-PDU ::= [3] IMPLICIT PDU

   -- [4] is obsolete

   GetBulkRequest-PDU ::= [5] IMPLICIT BulkPDU

   InformRequest-PDU ::= [6] IMPLICIT PDU

   SNMPv2-Trap-PDU ::= [7] IMPLICIT PDU

   --   Usage and precise semantics of Report-PDU are not defined
   --   in this document.  Any SNMP administrative framework making
   --   use of this PDU must define its usage and semantics.
```

```
   Report-PDU ::= [8] IMPLICIT PDU

   max-bindings INTEGER ::= 2147483647

   PDU ::= SEQUENCE {
           request-id INTEGER (-214783648..214783647),

           error-status                -- sometimes ignored
               INTEGER {
                   noError(0),
                   tooBig(1),
                   noSuchName(2),      -- for proxy compatibility
                   badValue(3),        -- for proxy compatibility
                   readOnly(4),        -- for proxy compatibility
                   genErr(5),
                   noAccess(6),
                   wrongType(7),
                   wrongLength(8),
                   wrongEncoding(9),
                   wrongValue(10),
                   noCreation(11),
                   inconsistentValue(12),
                   resourceUnavailable(13),
                   commitFailed(14),
                   undoFailed(15),
                   authorizationError(16),
                   notWritable(17),
                   inconsistentName(18)
               },

           error-index                 -- sometimes ignored
               INTEGER (0..max-bindings),

           variable-bindings           -- values are sometimes ignored
               VarBindList
       }

   BulkPDU ::=                         -- must be identical in
       SEQUENCE {                      -- structure to PDU
           request-id      INTEGER (-214783648..214783647),
           non-repeaters   INTEGER (0..max-bindings),
           max-repetitions INTEGER (0..max-bindings),

           variable-bindings           -- values are ignored
               VarBindList
       }

   -- variable binding
```

```
   VarBind ::= SEQUENCE {
           name ObjectName,

           CHOICE {
               value          ObjectSyntax,
               unSpecified    NULL,    -- in retrieval requests

                                       -- exceptions in responses
               noSuchObject   [0] IMPLICIT NULL,
               noSuchInstance [1] IMPLICIT NULL,
               endOfMibView   [2] IMPLICIT NULL
           }
       }

   -- variable-binding list

   VarBindList ::= SEQUENCE (SIZE (0..max-bindings)) OF VarBind

   END
```

4. Protocol Specification

4.1. Common Constructs

The value of the request-id field in a Response-PDU takes the value of the request-id field in the request PDU to which it is a response. By use of the request-id value, an application can distinguish the (potentially multiple) outstanding requests, and thereby correlate incoming responses with outstanding requests. In cases where an unreliable datagram service is used, the request-id also provides a simple means of identifying messages duplicated by the network. Use of the same request-id on a retransmission of a request allows the response to either the original transmission or the retransmission to satisfy the request. However, in order to calculate the round trip time for transmission and processing of a request-response transaction, the application needs to use a different request-id value on a retransmitted request. The latter strategy is recommended for use in the majority of situations.

A non-zero value of the error-status field in a Response-PDU is used to indicate that an error occurred to prevent the processing of the request. In these cases, a non-zero value of the Response-PDU's error-index field provides additional information by identifying which variable binding in the list caused the error. A variable binding is identified by its index value. The first variable binding in a variable-binding list is index one, the second is index two, etc.

SNMP limits OBJECT IDENTIFIER values to a maximum of 128 sub-identifiers, where each sub-identifier has a maximum value of 2**32-1.

4.2. PDU Processing

In the elements of procedure below, any field of a PDU which is not referenced by the relevant procedure is ignored by the receiving SNMP entity. However, all components of a PDU, including those whose values are ignored by the receiving SNMP entity, must have valid ASN.1 syntax and encoding. For example, some PDUs (e.g., the GetRequest-PDU) are concerned only with the name of a variable and not its value. In this case, the value portion of the variable binding is ignored by the receiving SNMP entity. The unSpecified value is defined for use as the value portion of such bindings.

On generating a management communication, the message "wrapper" to encapsulate the PDU is generated according to the "Elements of Procedure" of the administrative framework in use. The definition of "max-bindings" imposes an upper bound on the number of variable bindings. In practice, the size of a message is also limited by constraints on the maximum message size. A compliant implementation must support as many variable bindings in a PDU or BulkPDU as fit into the overall maximum message size limit of the SNMP engine, but no more than 2147483647 variable bindings.

On receiving a management communication, the "Elements of Procedure" of the administrative framework in use is followed, and if those procedures indicate that the operation contained within the message is to be performed locally, then those procedures also indicate the MIB view which is visible to the operation.

4.2.1. The GetRequest-PDU

A GetRequest-PDU is generated and transmitted at the request of an application.

Upon receipt of a GetRequest-PDU, the receiving SNMP entity processes each variable binding in the variable-binding list to produce a Response-PDU. All fields of the Response-PDU have the same values as the corresponding fields of the received request except as indicated below. Each variable binding is processed as follows:

(1) If the variable binding’s name exactly matches the name of a variable accessible by this request, then the variable binding’s value field is set to the value of the named variable.

(2) Otherwise, if the variable binding's name does not have an OBJECT IDENTIFIER prefix which exactly matches the OBJECT IDENTIFIER prefix of any (potential) variable accessible by this request, then its value field is set to "noSuchObject".

(3) Otherwise, the variable binding's value field is set to "noSuchInstance".

If the processing of any variable binding fails for a reason other than listed above, then the Response-PDU is re-formatted with the same values in its request-id and variable-bindings fields as the received GetRequest-PDU, with the value of its error-status field set to "genErr", and the value of its error-index field is set to the index of the failed variable binding.

Otherwise, the value of the Response-PDU's error-status field is set to "noError", and the value of its error-index field is zero.

The generated Response-PDU is then encapsulated into a message. If the size of the resultant message is less than or equal to both a local constraint and the maximum message size of the originator, it is transmitted to the originator of the GetRequest-PDU.

Otherwise, an alternate Response-PDU is generated. This alternate Response-PDU is formatted with the same value in its request-id field as the received GetRequest-PDU, with the value of its error-status field set to "tooBig", the value of its error-index field set to zero, and an empty variable-bindings field. This alternate Response-PDU is then encapsulated into a message. If the size of the resultant message is less than or equal to both a local constraint and the maximum message size of the originator, it is transmitted to the originator of the GetRequest-PDU. Otherwise, the snmpSilentDrops [RFC3418] counter is incremented and the resultant message is discarded.

4.2.2. The GetNextRequest-PDU

A GetNextRequest-PDU is generated and transmitted at the request of an application.

Upon receipt of a GetNextRequest-PDU, the receiving SNMP entity processes each variable binding in the variable-binding list to produce a Response-PDU. All fields of the Response-PDU have the same values as the corresponding fields of the received request except as indicated below. Each variable binding is processed as follows:

(1) The variable is located which is in the lexicographically ordered list of the names of all variables which are

accessible by this request and whose name is the first lexicographic successor of the variable binding's name in the incoming GetNextRequest-PDU. The corresponding variable binding's name and value fields in the Response-PDU are set to the name and value of the located variable.

(2) If the requested variable binding's name does not lexicographically precede the name of any variable accessible by this request, i.e., there is no lexicographic successor, then the corresponding variable binding produced in the Response-PDU has its value field set to "endOfMibView", and its name field set to the variable binding's name in the request.

If the processing of any variable binding fails for a reason other than listed above, then the Response-PDU is re-formatted with the same values in its request-id and variable-bindings fields as the received GetNextRequest-PDU, with the value of its error-status field set to "genErr", and the value of its error-index field is set to the index of the failed variable binding.

Otherwise, the value of the Response-PDU's error-status field is set to "noError", and the value of its error-index field is zero.

The generated Response-PDU is then encapsulated into a message. If the size of the resultant message is less than or equal to both a local constraint and the maximum message size of the originator, it is transmitted to the originator of the GetNextRequest-PDU.

Otherwise, an alternate Response-PDU is generated. This alternate Response-PDU is formatted with the same values in its request-id field as the received GetNextRequest-PDU, with the value of its error-status field set to "tooBig", the value of its error-index field set to zero, and an empty variable-bindings field. This alternate Response-PDU is then encapsulated into a message. If the size of the resultant message is less than or equal to both a local constraint and the maximum message size of the originator, it is transmitted to the originator of the GetNextRequest-PDU. Otherwise, the snmpSilentDrops [RFC3418] counter is incremented and the resultant message is discarded.

4.2.2.1. Example of Table Traversal

An important use of the GetNextRequest-PDU is the traversal of conceptual tables of information within a MIB. The semantics of this type of request, together with the method of identifying individual instances of objects in the MIB, provides access to related objects in the MIB as if they enjoyed a tabular organization.

In the protocol exchange sketched below, an application retrieves the media-dependent physical address and the address-mapping type for each entry in the IP net-to-media Address Translation Table [RFC1213] of a particular network element. It also retrieves the value of sysUpTime [RFC3418], at which the mappings existed. Suppose that the command responder's IP net-to-media table has three entries:

Interface-Number	Network-Address	Physical-Address	Type
1	10.0.0.51	00:00:10:01:23:45	static
1	9.2.3.4	00:00:10:54:32:10	dynamic
2	10.0.0.15	00:00:10:98:76:54	dynamic

The SNMP entity supporting a command generator application begins by sending a GetNextRequest-PDU containing the indicated OBJECT IDENTIFIER values as the requested variable names:

```
 GetNextRequest ( sysUpTime,
                  ipNetToMediaPhysAddress,
                  ipNetToMediaType )
```

The SNMP entity supporting a command responder application responds with a Response-PDU:

```
 Response (( sysUpTime.0 =  "123456" ),
             ( ipNetToMediaPhysAddress.1.9.2.3.4 = "000010543210" ),
          ( ipNetToMediaType.1.9.2.3.4 =  "dynamic" ))
```

The SNMP entity supporting the command generator application continues with:

```
 GetNextRequest ( sysUpTime,
                  ipNetToMediaPhysAddress.1.9.2.3.4,
                  ipNetToMediaType.1.9.2.3.4 )
```

The SNMP entity supporting the command responder application responds with:

```
 Response (( sysUpTime.0 =  "123461" ),
             ( ipNetToMediaPhysAddress.1.10.0.0.51 = "000010012345" ),
          ( ipNetToMediaType.1.10.0.0.51 =  "static" ))
```

The SNMP entity supporting the command generator application continues with:

```
 GetNextRequest ( sysUpTime,
                  ipNetToMediaPhysAddress.1.10.0.0.51,
                  ipNetToMediaType.1.10.0.0.51 )
```

The SNMP entity supporting the command responder application responds with:

```
 Response (( sysUpTime.0 =  "123466" ),
            ( ipNetToMediaPhysAddress.2.10.0.0.15 = "000010987654" ),
          ( ipNetToMediaType.2.10.0.0.15 =  "dynamic" ))
```

The SNMP entity supporting the command generator application continues with:

```
 GetNextRequest ( sysUpTime,
                  ipNetToMediaPhysAddress.2.10.0.0.15,
                  ipNetToMediaType.2.10.0.0.15 )
```

As there are no further entries in the table, the SNMP entity supporting the command responder application responds with the variables that are next in the lexicographical ordering of the accessible object names, for example:

```
 Response (( sysUpTime.0 =  "123471" ),
            ( ipNetToMediaNetAddress.1.9.2.3.4 = "9.2.3.4" ),
          ( ipRoutingDiscards.0 =  "2" ))
```

Note how, having reached the end of the column for ipNetToMediaPhysAddress, the second variable binding from the command responder application has now "wrapped" to the first row in the next column. Furthermore, note how, having reached the end of the ipNetToMediaTable for the third variable binding, the command responder application has responded with the next available object, which is outside that table. This response signals the end of the table to the command generator application.

4.2.3. The GetBulkRequest-PDU

A GetBulkRequest-PDU is generated and transmitted at the request of an application. The purpose of the GetBulkRequest-PDU is to request the transfer of a potentially large amount of data, including, but not limited to, the efficient and rapid retrieval of large tables.

Upon receipt of a GetBulkRequest-PDU, the receiving SNMP entity processes each variable binding in the variable-binding list to produce a Response-PDU with its request-id field having the same value as in the request.

For the GetBulkRequest-PDU type, the successful processing of each variable binding in the request generates zero or more variable bindings in the Response-PDU. That is, the one-to-one mapping between the variable bindings of the GetRequest-PDU, GetNextRequest-

PDU, and SetRequest-PDU types and the resultant Response-PDUs does not apply for the mapping between the variable bindings of a GetBulkRequest-PDU and the resultant Response-PDU.

The values of the non-repeaters and max-repetitions fields in the request specify the processing requested. One variable binding in the Response-PDU is requested for the first N variable bindings in the request and M variable bindings are requested for each of the R remaining variable bindings in the request. Consequently, the total number of requested variable bindings communicated by the request is given by N + (M * R), where N is the minimum of: a) the value of the non-repeaters field in the request, and b) the number of variable bindings in the request; M is the value of the max-repetitions field in the request; and R is the maximum of: a) number of variable bindings in the request - N, and b) zero.

The receiving SNMP entity produces a Response-PDU with up to the total number of requested variable bindings communicated by the request. The request-id shall have the same value as the received GetBulkRequest-PDU.

If N is greater than zero, the first through the (N)-th variable bindings of the Response-PDU are each produced as follows:

(1) The variable is located which is in the lexicographically ordered list of the names of all variables which are accessible by this request and whose name is the first lexicographic successor of the variable binding's name in the incoming GetBulkRequest-PDU. The corresponding variable binding's name and value fields in the Response-PDU are set to the name and value of the located variable.

(2) If the requested variable binding's name does not lexicographically precede the name of any variable accessible by this request, i.e., there is no lexicographic successor, then the corresponding variable binding produced in the Response-PDU has its value field set to "endOfMibView", and its name field set to the variable binding's name in the request.

If M and R are non-zero, the (N + 1)-th and subsequent variable bindings of the Response-PDU are each produced in a similar manner. For each iteration i, such that i is greater than zero and less than or equal to M, and for each repeated variable, r, such that r is greater than zero and less than or equal to R, the (N + ((i-1) * R) + r)-th variable binding of the Response-PDU is produced as follows:

(1) The variable which is in the lexicographically ordered list of the names of all variables which are accessible by this request and whose name is the (i)-th lexicographic successor of the (N + r)-th variable binding's name in the incoming GetBulkRequest-PDU is located and the variable binding's name and value fields are set to the name and value of the located variable.

(2) If there is no (i)-th lexicographic successor, then the corresponding variable binding produced in the Response-PDU has its value field set to "endOfMibView", and its name field set to either the last lexicographic successor, or if there are no lexicographic successors, to the (N + r)-th variable binding's name in the request.

While the maximum number of variable bindings in the Response-PDU is bounded by N + (M * R), the response may be generated with a lesser number of variable bindings (possibly zero) for either of three reasons.

(1) If the size of the message encapsulating the Response-PDU containing the requested number of variable bindings would be greater than either a local constraint or the maximum message size of the originator, then the response is generated with a lesser number of variable bindings. This lesser number is the ordered set of variable bindings with some of the variable bindings at the end of the set removed, such that the size of the message encapsulating the Response-PDU is approximately equal to but no greater than either a local constraint or the maximum message size of the originator. Note that the number of variable bindings removed has no relationship to the values of N, M, or R.

(2) The response may also be generated with a lesser number of variable bindings if for some value of iteration i, such that i is greater than zero and less than or equal to M, that all of the generated variable bindings have the value field set to "endOfMibView". In this case, the variable bindings may be truncated after the (N + (i * R))-th variable binding.

(3) In the event that the processing of a request with many repetitions requires a significantly greater amount of processing time than a normal request, then a command responder application may terminate the request with less than the full number of repetitions, providing at least one repetition is completed.

If the processing of any variable binding fails for a reason other than listed above, then the Response-PDU is re-formatted with the same values in its request-id and variable-bindings fields as the received GetBulkRequest-PDU, with the value of its error-status field set to "genErr", and the value of its error-index field is set to the index of the variable binding in the original request which corresponds to the failed variable binding.

Otherwise, the value of the Response-PDU's error-status field is set to "noError", and the value of its error-index field to zero.

The generated Response-PDU (possibly with an empty variable-bindings field) is then encapsulated into a message. If the size of the resultant message is less than or equal to both a local constraint and the maximum message size of the originator, it is transmitted to the originator of the GetBulkRequest-PDU. Otherwise, the snmpSilentDrops [RFC3418] counter is incremented and the resultant message is discarded.

4.2.3.1. Another Example of Table Traversal

This example demonstrates how the GetBulkRequest-PDU can be used as an alternative to the GetNextRequest-PDU. The same traversal of the IP net-to-media table as shown in Section 4.2.2.1 is achieved with fewer exchanges.

The SNMP entity supporting the command generator application begins by sending a GetBulkRequest-PDU with the modest max-repetitions value of 2, and containing the indicated OBJECT IDENTIFIER values as the requested variable names:

```
 GetBulkRequest [ non-repeaters = 1, max-repetitions = 2 ]
                ( sysUpTime,
                  ipNetToMediaPhysAddress,
                  ipNetToMediaType )
```

The SNMP entity supporting the command responder application responds with a Response-PDU:

```
 Response (( sysUpTime.0 =  "123456" ),
             ( ipNetToMediaPhysAddress.1.9.2.3.4 = "000010543210" ),
          ( ipNetToMediaType.1.9.2.3.4 =  "dynamic" ),
             ( ipNetToMediaPhysAddress.1.10.0.0.51 = "000010012345" ),
          ( ipNetToMediaType.1.10.0.0.51 =  "static" ))
```

The SNMP entity supporting the command generator application continues with:

```
  GetBulkRequest [ non-repeaters = 1, max-repetitions = 2 ]
                 ( sysUpTime,
                   ipNetToMediaPhysAddress.1.10.0.0.51,
                   ipNetToMediaType.1.10.0.0.51 )
```

The SNMP entity supporting the command responder application responds with:

```
 Response (( sysUpTime.0 =  "123466" ),
            ( ipNetToMediaPhysAddress.2.10.0.0.15 = "000010987654" ),
            ( ipNetToMediaType.2.10.0.0.15 = "dynamic" ),
            ( ipNetToMediaNetAddress.1.9.2.3.4 = "9.2.3.4" ),
          ( ipRoutingDiscards.0 =  "2" ))
```

Note how, as in the first example, the variable bindings in the response indicate that the end of the table has been reached. The fourth variable binding does so by returning information from the next available column; the fifth variable binding does so by returning information from the first available object lexicographically following the table. This response signals the end of the table to the command generator application.

4.2.4. The Response-PDU

The Response-PDU is generated by an SNMP entity only upon receipt of a GetRequest-PDU, GetNextRequest-PDU, GetBulkRequest-PDU, SetRequest-PDU, or InformRequest-PDU, as described elsewhere in this document.

If the error-status field of the Response-PDU is non-zero, the value fields of the variable bindings in the variable binding list are ignored.

If both the error-status field and the error-index field of the Response-PDU are non-zero, then the value of the error-index field is the index of the variable binding (in the variable-binding list of the corresponding request) for which the request failed. The first variable binding in a request's variable-binding list is index one, the second is index two, etc.

A compliant SNMP entity supporting a command generator application must be able to properly receive and handle a Response-PDU with an error-status field equal to "noSuchName", "badValue", or "readOnly". (See sections 1.3 and 4.3 of [RFC2576].)

Upon receipt of a Response-PDU, the receiving SNMP entity presents its contents to the application which generated the request with the same request-id value. For more details, see [RFC3412].

4.2.5. The SetRequest-PDU

A SetRequest-PDU is generated and transmitted at the request of an application.

Upon receipt of a SetRequest-PDU, the receiving SNMP entity determines the size of a message encapsulating a Response-PDU having the same values in its request-id and variable-bindings fields as the received SetRequest-PDU, and the largest possible sizes of the error-status and error-index fields. If the determined message size is greater than either a local constraint or the maximum message size of the originator, then an alternate Response-PDU is generated, transmitted to the originator of the SetRequest-PDU, and processing of the SetRequest-PDU terminates immediately thereafter. This alternate Response-PDU is formatted with the same values in its request-id field as the received SetRequest-PDU, with the value of its error-status field set to "tooBig", the value of its error-index field set to zero, and an empty variable-bindings field. This alternate Response-PDU is then encapsulated into a message. If the size of the resultant message is less than or equal to both a local constraint and the maximum message size of the originator, it is transmitted to the originator of the SetRequest-PDU. Otherwise, the snmpSilentDrops [RFC3418] counter is incremented and the resultant message is discarded. Regardless, processing of the SetRequest-PDU terminates.

Otherwise, the receiving SNMP entity processes each variable binding in the variable-binding list to produce a Response-PDU. All fields of the Response-PDU have the same values as the corresponding fields of the received request except as indicated below.

The variable bindings are conceptually processed as a two phase operation. In the first phase, each variable binding is validated; if all validations are successful, then each variable is altered in the second phase. Of course, implementors are at liberty to implement either the first, or second, or both, of these conceptual phases as multiple implementation phases. Indeed, such multiple implementation phases may be necessary in some cases to ensure consistency.

The following validations are performed in the first phase on each variable binding until they are all successful, or until one fails:

(1) If the variable binding's name specifies an existing or non-existent variable to which this request is/would be denied access because it is/would not be in the appropriate MIB view, then the value of the Response-PDU's error-status field is set to "noAccess", and the value of its error-index field is set to the index of the failed variable binding.

(2) Otherwise, if there are no variables which share the same OBJECT IDENTIFIER prefix as the variable binding's name, and which are able to be created or modified no matter what new value is specified, then the value of the Response-PDU's error-status field is set to "notWritable", and the value of its error-index field is set to the index of the failed variable binding.

(3) Otherwise, if the variable binding's value field specifies, according to the ASN.1 language, a type which is inconsistent with that required for all variables which share the same OBJECT IDENTIFIER prefix as the variable binding's name, then the value of the Response-PDU's error-status field is set to "wrongType", and the value of its error-index field is set to the index of the failed variable binding.

(4) Otherwise, if the variable binding's value field specifies, according to the ASN.1 language, a length which is inconsistent with that required for all variables which share the same OBJECT IDENTIFIER prefix as the variable binding's name, then the value of the Response-PDU's error-status field is set to "wrongLength", and the value of its error-index field is set to the index of the failed variable binding.

(5) Otherwise, if the variable binding's value field contains an ASN.1 encoding which is inconsistent with that field's ASN.1 tag, then the value of the Response-PDU's error-status field is set to "wrongEncoding", and the value of its error-index field is set to the index of the failed variable binding. (Note that not all implementation strategies will generate this error.)

(6) Otherwise, if the variable binding's value field specifies a value which could under no circumstances be assigned to the variable, then the value of the Response-PDU's error-status field is set to "wrongValue", and the value of its error-index field is set to the index of the failed variable binding.

(7) Otherwise, if the variable binding's name specifies a variable which does not exist and could not ever be created (even though some variables sharing the same OBJECT IDENTIFIER prefix might under some circumstances be able to be created), then the value of the Response-PDU's error-status field is set to "noCreation", and the value of its error-index field is set to the index of the failed variable binding.

(8) Otherwise, if the variable binding's name specifies a variable which does not exist but can not be created under the present circumstances (even though it could be created under other circumstances), then the value of the Response-PDU's error-status field is set to "inconsistentName", and the value of its error-index field is set to the index of the failed variable binding.

(9) Otherwise, if the variable binding's name specifies a variable which exists but can not be modified no matter what new value is specified, then the value of the Response-PDU's error-status field is set to "notWritable", and the value of its error-index field is set to the index of the failed variable binding.

(10) Otherwise, if the variable binding's value field specifies a value that could under other circumstances be held by the variable, but is presently inconsistent or otherwise unable to be assigned to the variable, then the value of the Response-PDU's error-status field is set to "inconsistentValue", and the value of its error-index field is set to the index of the failed variable binding.

(11) When, during the above steps, the assignment of the value specified by the variable binding's value field to the specified variable requires the allocation of a resource which is presently unavailable, then the value of the Response-PDU's error-status field is set to "resourceUnavailable", and the value of its error-index field is set to the index of the failed variable binding.

(12) If the processing of the variable binding fails for a reason other than listed above, then the value of the Response-PDU's error-status field is set to "genErr", and the value of its error-index field is set to the index of the failed variable binding.

(13) Otherwise, the validation of the variable binding succeeds.

At the end of the first phase, if the validation of all variable bindings succeeded, then the value of the Response-PDU's error-status field is set to "noError" and the value of its error-index field is zero, and processing continues as follows.

For each variable binding in the request, the named variable is created if necessary, and the specified value is assigned to it. Each of these variable assignments occurs as if simultaneously with respect to all other assignments specified in the same request. However, if the same variable is named more than once in a single request, with different associated values, then the actual assignment made to that variable is implementation-specific.

If any of these assignments fail (even after all the previous validations), then all other assignments are undone, and the Response-PDU is modified to have the value of its error-status field set to "commitFailed", and the value of its error-index field set to the index of the failed variable binding.

If and only if it is not possible to undo all the assignments, then the Response-PDU is modified to have the value of its error-status field set to "undoFailed", and the value of its error-index field is set to zero. Note that implementations are strongly encouraged to take all possible measures to avoid use of either "commitFailed" or "undoFailed" - these two error-status codes are not to be taken as license to take the easy way out in an implementation.

Finally, the generated Response-PDU is encapsulated into a message, and transmitted to the originator of the SetRequest-PDU.

4.2.6. The SNMPv2-Trap-PDU

An SNMPv2-Trap-PDU is generated and transmitted by an SNMP entity on behalf of a notification originator application. The SNMPv2-Trap-PDU is often used to notify a notification receiver application at a logically remote SNMP entity that an event has occurred or that a condition is present. There is no confirmation associated with this notification delivery mechanism.

The destination(s) to which an SNMPv2-Trap-PDU is sent is determined in an implementation-dependent fashion by the SNMP entity. The first two variable bindings in the variable binding list of an SNMPv2-Trap-PDU are sysUpTime.0 [RFC3418] and snmpTrapOID.0 [RFC3418] respectively. If the OBJECTS clause is present in the invocation of the corresponding NOTIFICATION-TYPE macro, then each corresponding variable, as instantiated by this notification, is copied, in order,

to the variable-bindings field. If any additional variables are being included (at the option of the generating SNMP entity), then each is copied to the variable-bindings field.

4.2.7. The InformRequest-PDU

An InformRequest-PDU is generated and transmitted by an SNMP entity on behalf of a notification originator application. The InformRequest-PDU is often used to notify a notification receiver application that an event has occurred or that a condition is present. This is a confirmed notification delivery mechanism, although there is, of course, no guarantee of delivery.

The destination(s) to which an InformRequest-PDU is sent is specified by the notification originator application. The first two variable bindings in the variable binding list of an InformRequest-PDU are sysUpTime.0 [RFC3418] and snmpTrapOID.0 [RFC3418] respectively. If the OBJECTS clause is present in the invocation of the corresponding NOTIFICATION-TYPE macro, then each corresponding variable, as instantiated by this notification, is copied, in order, to the variable-bindings field. If any additional variables are being included (at the option of the generating SNMP entity), then each is copied to the variable-bindings field.

Upon receipt of an InformRequest-PDU, the receiving SNMP entity determines the size of a message encapsulating a Response-PDU with the same values in its request-id, error-status, error-index and variable-bindings fields as the received InformRequest-PDU. If the determined message size is greater than either a local constraint or the maximum message size of the originator, then an alternate Response-PDU is generated, transmitted to the originator of the InformRequest-PDU, and processing of the InformRequest-PDU terminates immediately thereafter. This alternate Response-PDU is formatted with the same values in its request-id field as the received InformRequest-PDU, with the value of its error-status field set to "tooBig", the value of its error-index field set to zero, and an empty variable-bindings field. This alternate Response-PDU is then encapsulated into a message. If the size of the resultant message is less than or equal to both a local constraint and the maximum message size of the originator, it is transmitted to the originator of the InformRequest-PDU. Otherwise, the snmpSilentDrops [RFC3418] counter is incremented and the resultant message is discarded. Regardless, processing of the InformRequest-PDU terminates.

Otherwise, the receiving SNMP entity:

(1) presents its contents to the appropriate application;

(2) generates a Response-PDU with the same values in its request-id and variable-bindings fields as the received InformRequest-PDU, with the value of its error-status field set to "noError" and the value of its error-index field set to zero; and

(3) transmits the generated Response-PDU to the originator of the InformRequest-PDU.

5. Notice on Intellectual Property

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-related documentation can be found in BCP-11. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to 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.

6. Acknowledgments

This document is the product of the SNMPv3 Working Group. Some special thanks are in order to the following Working Group members:

Randy Bush
Jeffrey D. Case
Mike Daniele
Rob Frye
Lauren Heintz
Keith McCloghrie
Russ Mundy
David T. Perkins
Randy Presuhn
Aleksey Romanov
Juergen Schoenwaelder
Bert Wijnen

This version of the document, edited by Randy Presuhn, was initially based on the work of a design team whose members were:

Jeffrey D. Case
Keith McCloghrie
David T. Perkins
Randy Presuhn
Juergen Schoenwaelder

The previous versions of this document, edited by Keith McCloghrie, was the result of significant work by four major contributors:

Jeffrey D. Case
Keith McCloghrie
Marshall T. Rose
Steven Waldbusser

Additionally, the contributions of the SNMPv2 Working Group to the previous versions are also acknowledged. In particular, a special thanks is extended for the contributions of:

Alexander I. Alten
Dave Arneson
Uri Blumenthal
Doug Book
Kim Curran
Jim Galvin
Maria Greene
Iain Hanson
Dave Harrington
Nguyen Hien
Jeff Johnson
Michael Kornegay
Deirdre Kostick
David Levi
Daniel Mahoney
Bob Natale
Brian O'Keefe
Andrew Pearson
Dave Perkins
Randy Presuhn
Aleksey Romanov
Shawn Routhier
Jon Saperia
Juergen Schoenwaelder
Bob Stewart

Kaj Tesink
Glenn Waters
Bert Wijnen

7. Security Considerations

The protocol defined in this document by itself does not provide a secure environment. Even if the network itself is secure (for example by using IPSec), there is no control as to who on the secure network is allowed access to management information.

It is recommended that the implementors consider the security features as provided by the SNMPv3 framework. Specifically, the use of the User-based Security Model STD 62, RFC 3414 [RFC3414] and the View-based Access Control Model STD 62, RFC 3415 [RFC3415] is recommended.

It is then a customer/user responsibility to ensure that the SNMP entity is properly configured so that:

- only those principals (users) having legitimate rights can access or modify the values of any MIB objects supported by that entity;

- the occurrence of particular events on the entity will be communicated appropriately;

- the entity responds appropriately and with due credence to events and information that have been communicated to it.

8. References

8.1. Normative References

[RFC768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980.

[RFC2578] McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M. and S. Waldbusser, "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, April 1999.

[RFC2579] McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M. and S. Waldbusser, "Textual Conventions for SMIv2", STD 58, RFC 2579, April 1999.

[RFC2580] McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M. and S. Waldbusser, "Conformance Statements for SMIv2", STD 58, RFC 2580, April 1999.

[RFC3411] Harrington, D., Presuhn, R. and B. Wijnen, "An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks", STD 62, RFC 3411, December 2002.

[RFC3412] 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.

[RFC3413] Levi, D., Meyer, P. and B. Stewart, "Simple Network Management Protocol (SNMP) Applications", STD 62, RFC 3413, December 2002.

[RFC3414] Blumenthal, U. and B. Wijnen, "The User-Based Security Model (USM) for Version 3 of the Simple Network Management Protocol (SNMPv3)", STD 62, RFC 3414, December 2002.

[RFC3415] 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.

[RFC3417] Presuhn, R., Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Transport Mappings for the Simple Network Management Protocol", STD 62, RFC 3417, December 2002.

[RFC3418] Presuhn, R., Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Management Information Base (MIB) for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3418, December 2002.

[ASN1] Information processing systems - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1), International Organization for Standardization. International Standard 8824, December 1987.

8.2. Informative References

[FRAG] Kent, C. and J. Mogul, "Fragmentation Considered Harmful," Proceedings, ACM SIGCOMM '87, Stowe, VT, August 1987.

[RFC1155] Rose, M. and K. McCloghrie, "Structure and Identification of Management Information for TCP/IP-based Internets", STD 16, RFC 1155, May 1990.

[RFC1157] Case, J., Fedor, M., Schoffstall, M. and J. Davin, "Simple Network Management Protocol", STD 15, RFC 1157, May 1990.

[RFC1212] Rose, M. and K. McCloghrie, "Concise MIB Definitions", STD 16, RFC 1212, March 1991.

[RFC1213] McCloghrie, K. and M. Rose, Editors, "Management Information Base for Network Management of TCP/IP-based internets: MIB-II", STD 17, RFC 1213, March 1991.

[RFC1215] Rose, M., "A Convention for Defining Traps for use with the SNMP", RFC 1215, March 1991.

[RFC1901] Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Introduction to Community-based SNMPv2", RFC 1901, January 1996.

[RFC2576] Frye, R., Levi, D., Routhier, S. and B. Wijnen, "Coexistence between Version 1, Version 2, and Version 3 of the Internet-Standard Network Management Framework", RFC 2576, March 2000.

[RFC2863] McCloghrie, K. and F. Kastenholz, "The Interfaces Group MIB", RFC 2863, June 2000.

[RFC2914] Floyd, S., "Congestion Control Principles", BCP 41, RFC 2914, September 2000.

[RFC3410] Case, J., Mundy, R., Partain, D. and B. Stewart, "Introduction and Applicability Statements for Internet-Standard Management Framework", RFC 3410, December 2002.

9. Changes from RFC 1905

These are the changes from RFC 1905:

- Corrected spelling error in copyright statement;
- Updated copyright date;
- Updated with new editor’s name and contact information;
- Added notice on intellectual property;

- Cosmetic fixes to layout and typography;

- Added table of contents;

- Title changed;

- Updated document headers and footers;

- Deleted the old clause 2.3, entitled "Access to Management Information";

- Changed the way in which request-id was defined, though with the same ultimate syntax and semantics, to avoid coupling with SMI. This does not affect the protocol in any way;

- Replaced the word "exception" with the word "error" in the old clause 4.1. This does not affect the protocol in any way;

- Deleted the first two paragraphs of the old clause 4.2;

- Clarified the maximum number of variable bindings that an implementation must support in a PDU. This does not affect the protocol in any way;

- Replaced occurrences of "SNMPv2 application" with "application";

- Deleted three sentences in old clause 4.2.3 describing the handling of an impossible situation. This does not affect the protocol in any way;

- Clarified the use of the SNMPv2-Trap-Pdu in the old clause 4.2.6. This does not affect the protocol in any way;

- Aligned description of the use of the InformRequest-Pdu in old clause 4.2.7 with the architecture. This does not affect the protocol in any way;

- Updated references;

- Re-wrote introduction clause;

- Replaced manager/agent/SNMPv2 entity terminology with terminology from RFC 2571. This does not affect the protocol in any way;

- Eliminated IMPORTS from the SMI, replaced with equivalent in-line ASN.1. This does not affect the protocol in any way;

- Added notes calling attention to two different manifestations of reaching the end of a table in the table walk examples;

- Added content to security considerations clause;

- Updated ASN.1 comment on use of Report-PDU. This does not affect the protocol in any way;

- Updated acknowledgments section;

- Included information on handling of BITS;

- Deleted spurious comma in ASN.1 definition of PDUs;

- Added abstract;

- Made handling of additional variable bindings in informs consistent with that for traps. This was a correction of an editorial oversight, and reflects implementation practice;

- Added reference to RFC 2914.

10. Editor's Address

Randy Presuhn
BMC Software, Inc.
2141 North First Street
San Jose, CA 95131
USA

Phone: +1 408 546 1006
EMail: randy_presuhn@bmc.com

11. Full Copyright Statement

Acknowledgement

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

Network Working Group
Request for Comments: 3417
STD: 62
Obsoletes: 1906
Category: Standards Track

Editor of this version:
R. Presuhn
BMC Software, Inc.
Authors of previous version:
J. Case
SNMP Research, Inc.
K. McCloghrie
Cisco Systems, Inc.
M. Rose
Dover Beach Consulting, Inc.
S. Waldbusser
International Network Services
December 2002

Transport Mappings for the Simple Network Management Protocol (SNMP)

Status of this Memo

Copyright Notice

Abstract

This document defines the transport of Simple Network Management Protocol (SNMP) messages over various protocols. This document obsoletes RFC 1906.

Table of Contents

1. Introduction

For a detailed overview of the documents that describe the current Internet-Standard Management Framework, please refer to section 7 of RFC 3410 [RFC3410].

Managed objects are accessed via a virtual information store, termed the Management Information Base or MIB. MIB objects are generally accessed through the Simple Network Management Protocol (SNMP). Objects in the MIB are defined using the mechanisms defined in the Structure of Management Information (SMI). This memo specifies a MIB

module that is compliant to the SMIv2, which is described in STD 58, RFC 2578 [RFC2578], STD 58, RFC 2579 [RFC2579] and STD 58, RFC 2580 [RFC2580].

This document, Transport Mappings for the Simple Network Management Protocol, defines how the management protocol [RFC3416] may be carried over a variety of protocol suites. It is the purpose of this document to define how the SNMP maps onto an initial set of transport domains. At the time of this writing, work was in progress to define an IPv6 mapping, described in [RFC3419]. Other mappings may be defined in the future.

Although several mappings are defined, the mapping onto UDP over IPv4 is the preferred mapping for systems supporting IPv4. Systems implementing IPv4 MUST implement the mapping onto UDP over IPv4. To maximize interoperability, systems supporting other mappings SHOULD also provide for access via the UDP over IPv4 mapping.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, RFC 2119 [RFC2119].

2. Definitions

```
SNMPv2-TM DEFINITIONS ::= BEGIN

IMPORTS
    MODULE-IDENTITY, OBJECT-IDENTITY,
    snmpModules, snmpDomains, snmpProxys
        FROM SNMPv2-SMI
    TEXTUAL-CONVENTION
        FROM SNMPv2-TC;

snmpv2tm MODULE-IDENTITY
    LAST-UPDATED "200210160000Z"
    ORGANIZATION "IETF SNMPv3 Working Group"
    CONTACT-INFO
            "WG-EMail:   snmpv3@lists.tislabs.com
             Subscribe:  snmpv3-request@lists.tislabs.com

             Co-Chair:   Russ Mundy
                         Network Associates Laboratories
             postal:     15204 Omega Drive, Suite 300
                         Rockville, MD 20850-4601
                         USA
             EMail:      mundy@tislabs.com
             phone:      +1 301 947-7107
```

```
                Co-Chair:   David Harrington
                            Enterasys Networks
                postal:     35 Industrial Way
                            P. O. Box 5005
                            Rochester, NH 03866-5005
                            USA
                EMail:      dbh@enterasys.com
                phone:      +1 603 337-2614

                Editor:     Randy Presuhn
                            BMC Software, Inc.
                postal:     2141 North First Street
                            San Jose, CA 95131
                            USA
                EMail:      randy_presuhn@bmc.com
                phone:      +1 408 546-1006"
        DESCRIPTION
                "The MIB module for SNMP transport mappings.

                 Copyright (C) The Internet Society (2002). This
                 version of this MIB module is part of RFC 3417;
                 see the RFC itself for full legal notices.
                "
        REVISION     "200210160000Z"
        DESCRIPTION
                "Clarifications, published as RFC 3417."
        REVISION    "199601010000Z"
        DESCRIPTION
                "Clarifications, published as RFC 1906."
        REVISION    "199304010000Z"
        DESCRIPTION
                "The initial version, published as RFC 1449."
        ::= { snmpModules 19 }

    -- SNMP over UDP over IPv4

    snmpUDPDomain  OBJECT-IDENTITY
        STATUS     current
        DESCRIPTION
                "The SNMP over UDP over IPv4 transport domain.
                The corresponding transport address is of type
                SnmpUDPAddress."
        ::= { snmpDomains 1 }
```

```
   SnmpUDPAddress ::= TEXTUAL-CONVENTION
       DISPLAY-HINT "1d.1d.1d.1d/2d"
       STATUS       current
       DESCRIPTION
               "Represents a UDP over IPv4 address:

                  octets   contents        encoding
                   1-4     IP-address      network-byte order
                   5-6     UDP-port        network-byte order
               "
       SYNTAX       OCTET STRING (SIZE (6))

   -- SNMP over OSI

   snmpCLNSDomain OBJECT-IDENTITY
       STATUS     current
       DESCRIPTION
               "The SNMP over CLNS transport domain.
               The corresponding transport address is of type
               SnmpOSIAddress."
       ::= { snmpDomains 2 }

   snmpCONSDomain OBJECT-IDENTITY
       STATUS     current
       DESCRIPTION
               "The SNMP over CONS transport domain.
               The corresponding transport address is of type
               SnmpOSIAddress."
       ::= { snmpDomains 3 }

   SnmpOSIAddress ::= TEXTUAL-CONVENTION
       DISPLAY-HINT "*1x:/1x:"
       STATUS       current
       DESCRIPTION
               "Represents an OSI transport-address:

             octets   contents           encoding
                1     length of NSAP     'n' as an unsigned-integer
                                           (either 0 or from 3 to 20)
             2..(n+1) NSAP                concrete binary representation
             (n+2)..m TSEL                string of (up to 64) octets
               "
       SYNTAX       OCTET STRING (SIZE (1 | 4..85))
```

```
   -- SNMP over DDP

   snmpDDPDomain  OBJECT-IDENTITY
       STATUS     current
       DESCRIPTION
               "The SNMP over DDP transport domain.  The corresponding
               transport address is of type SnmpNBPAddress."
       ::= { snmpDomains 4 }

   SnmpNBPAddress ::= TEXTUAL-CONVENTION
       STATUS       current
       DESCRIPTION
               "Represents an NBP name:

            octets        contents          encoding
               1          length of object  'n' as an unsigned integer
             2..(n+1)     object            string of (up to 32) octets
              n+2         length of type    'p' as an unsigned integer
         (n+3)..(n+2+p)   type              string of (up to 32) octets
             n+3+p        length of zone    'q' as an unsigned integer
       (n+4+p)..(n+3+p+q) zone              string of (up to 32) octets

               For comparison purposes, strings are
               case-insensitive. All strings may contain any octet
               other than 255 (hex ff)."
       SYNTAX       OCTET STRING (SIZE (3..99))

   -- SNMP over IPX

   snmpIPXDomain  OBJECT-IDENTITY
       STATUS     current
       DESCRIPTION
               "The SNMP over IPX transport domain.  The corresponding
               transport address is of type SnmpIPXAddress."
       ::= { snmpDomains 5 }

   SnmpIPXAddress ::= TEXTUAL-CONVENTION
       DISPLAY-HINT "4x.1x:1x:1x:1x:1x:1x.2d"
       STATUS       current
       DESCRIPTION
               "Represents an IPX address:

                  octets   contents            encoding
                   1-4     network-number      network-byte order
                   5-10    physical-address    network-byte order
                  11-12    socket-number       network-byte order
               "
       SYNTAX       OCTET STRING (SIZE (12))
```

```
   -- for proxy to SNMPv1 (RFC 1157)

   rfc1157Proxy   OBJECT IDENTIFIER ::= { snmpProxys 1 }

   rfc1157Domain  OBJECT-IDENTITY
       STATUS     deprecated
       DESCRIPTION
               "The transport domain for SNMPv1 over UDP over IPv4.
               The corresponding transport address is of type
               SnmpUDPAddress."
       ::= { rfc1157Proxy 1 }

   --  ::= { rfc1157Proxy 2 }            this OID is obsolete

   END
```

3. SNMP over UDP over IPv4

This is the preferred transport mapping.

3.1. Serialization

Each instance of a message is serialized (i.e., encoded according to the convention of [BER]) onto a single UDP [RFC768] over IPv4 [RFC791] datagram, using the algorithm specified in Section 8.

3.2. Well-known Values

It is suggested that administrators configure their SNMP entities supporting command responder applications to listen on UDP port 161. Further, it is suggested that SNMP entities supporting notification receiver applications be configured to listen on UDP port 162.

When an SNMP entity uses this transport mapping, it must be capable of accepting messages up to and including 484 octets in size. It is recommended that implementations be capable of accepting messages of up to 1472 octets in size. Implementation of larger values is encouraged whenever possible.

4. SNMP over OSI

This is an optional transport mapping.

4.1. Serialization

Each instance of a message is serialized onto a single TSDU [IS8072] [IS8072A] for the OSI Connectionless-mode Transport Service (CLTS), using the algorithm specified in Section 8.

4.2. Well-known Values

It is suggested that administrators configure their SNMP entities supporting command responder applications to listen on transport selector "snmp-l" (which consists of six ASCII characters), when using a CL-mode network service to realize the CLTS. Further, it is suggested that SNMP entities supporting notification receiver applications be configured to listen on transport selector "snmpt-l" (which consists of seven ASCII characters, six letters and a hyphen) when using a CL-mode network service to realize the CLTS. Similarly, when using a CO-mode network service to realize the CLTS, the suggested transport selectors are "snmp-o" and "snmpt-o", for command responders and notification receivers, respectively.

When an SNMP entity uses this transport mapping, it must be capable of accepting messages that are at least 484 octets in size. Implementation of larger values is encouraged whenever possible.

5. SNMP over DDP

This is an optional transport mapping.

5.1. Serialization

Each instance of a message is serialized onto a single DDP datagram [APPLETALK], using the algorithm specified in Section 8.

5.2. Well-known Values

SNMP messages are sent using DDP protocol type 8. SNMP entities supporting command responder applications listen on DDP socket number 8, while SNMP entities supporting notification receiver applications listen on DDP socket number 9.

Administrators must configure their SNMP entities supporting command responder applications to use NBP type "SNMP Agent" (which consists of ten ASCII characters) while those supporting notification receiver applications must be configured to use NBP type "SNMP Trap Handler" (which consists of seventeen ASCII characters).

The NBP name for SNMP entities supporting command responders and notification receivers should be stable - NBP names should not change any more often than the IP address of a typical TCP/IP node. It is suggested that the NBP name be stored in some form of stable storage.

When an SNMP entity uses this transport mapping, it must be capable of accepting messages that are at least 484 octets in size. Implementation of larger values is encouraged whenever possible.

5.3. Discussion of AppleTalk Addressing

The AppleTalk protocol suite has certain features not manifest in the TCP/IP suite. AppleTalk's naming strategy and the dynamic nature of address assignment can cause problems for SNMP entities that wish to manage AppleTalk networks. TCP/IP nodes have an associated IP address which distinguishes each from the other. In contrast, AppleTalk nodes generally have no such characteristic. The network-level address, while often relatively stable, can change at every reboot (or more frequently).

Thus, when SNMP is mapped over DDP, nodes are identified by a "name", rather than by an "address". Hence, all AppleTalk nodes that implement this mapping are required to respond to NBP lookups and confirms (e.g., implement the NBP protocol stub), which guarantees that a mapping from NBP name to DDP address will be possible.

In determining the SNMP identity to register for an SNMP entity, it is suggested that the SNMP identity be a name which is associated with other network services offered by the machine.

NBP lookups, which are used to map NBP names into DDP addresses, can cause large amounts of network traffic as well as consume CPU resources. It is also the case that the ability to perform an NBP lookup is sensitive to certain network disruptions (such as zone table inconsistencies) which would not prevent direct AppleTalk communications between two SNMP entities.

Thus, it is recommended that NBP lookups be used infrequently, primarily to create a cache of name-to-address mappings. These cached mappings should then be used for any further SNMP traffic. It is recommended that SNMP entities supporting command generator applications should maintain this cache between reboots. This caching can help minimize network traffic, reduce CPU load on the network, and allow for (some amount of) network trouble shooting when the basic name-to-address translation mechanism is broken.

5.3.1. How to Acquire NBP names

An SNMP entity supporting command generator applications may have a pre-configured list of names of "known" SNMP entities supporting command responder applications. Similarly, an SNMP entity supporting command generator or notification receiver applications might interact with an operator. Finally, an SNMP entity supporting command generator or notification receiver applications might communicate with all SNMP entities supporting command responder or notification originator applications in a set of zones or networks.

5.3.2. When to Turn NBP names into DDP addresses

When an SNMP entity uses a cache entry to address an SNMP packet, it should attempt to confirm the validity mapping, if the mapping hasn't been confirmed within the last T1 seconds. This cache entry lifetime, T1, has a minimum, default value of 60 seconds, and should be configurable.

An SNMP entity supporting a command generator application may decide to prime its cache of names prior to actually communicating with another SNMP entity. In general, it is expected that such an entity may want to keep certain mappings "more current" than other mappings, e.g., those nodes which represent the network infrastructure (e.g., routers) may be deemed "more important".

Note that an SNMP entity supporting command generator applications should not prime its entire cache upon initialization - rather, it should attempt resolutions over an extended period of time (perhaps in some pre-determined or configured priority order). Each of these resolutions might, in fact, be a wildcard lookup in a given zone.

An SNMP entity supporting command responder applications must never prime its cache. When generating a response, such an entity does not need to confirm a cache entry. An SNMP entity supporting notification originator applications should do NBP lookups (or confirms) only when it needs to send an SNMP trap or inform.

5.3.3. How to Turn NBP names into DDP addresses

If the only piece of information available is the NBP name, then an NBP lookup should be performed to turn that name into a DDP address. However, if there is a piece of stale information, it can be used as a hint to perform an NBP confirm (which sends a unicast to the network address which is presumed to be the target of the name lookup) to see if the stale information is, in fact, still valid.

An NBP name to DDP address mapping can also be confirmed implicitly using only SNMP transactions. For example, an SNMP entity supporting command generator applications issuing a retrieval operation could also retrieve the relevant objects from the NBP group [RFC1742] for the SNMP entity supporting the command responder application. This information can then be correlated with the source DDP address of the response.

5.3.4. What if NBP is broken

Under some circumstances, there may be connectivity between two SNMP entities, but the NBP mapping machinery may be broken, e.g.,

- the NBP FwdReq (forward NBP lookup onto local attached network) mechanism might be broken at a router on the other entity's network; or,

- the NBP BrRq (NBP broadcast request) mechanism might be broken at a router on the entity's own network; or,

- NBP might be broken on the other entity's node.

An SNMP entity supporting command generator applications which is dedicated to AppleTalk management might choose to alleviate some of these failures by directly implementing the router portion of NBP. For example, such an entity might already know all the zones on the AppleTalk internet and the networks on which each zone appears. Given an NBP lookup which fails, the entity could send an NBP FwdReq to the network in which the SNMP entity supporting the command responder or notification originator application was last located. If that failed, the station could then send an NBP LkUp (NBP lookup packet) as a directed (DDP) multicast to each network number on that network. Of the above (single) failures, this combined approach will solve the case where either the local router's BrRq-to-FwdReq mechanism is broken or the remote router's FwdReq-to-LkUp mechanism is broken.

6. SNMP over IPX

This is an optional transport mapping.

6.1. Serialization

Each instance of a message is serialized onto a single IPX datagram [NOVELL], using the algorithm specified in Section 8.

6.2. Well-known Values

SNMP messages are sent using IPX packet type 4 (i.e., Packet Exchange Protocol).

It is suggested that administrators configure their SNMP entities supporting command responder applications to listen on IPX socket 36879 (900f hexadecimal). Further, it is suggested that those supporting notification receiver applications be configured to listen on IPX socket 36880 (9010 hexadecimal).

When an SNMP entity uses this transport mapping, it must be capable of accepting messages that are at least 546 octets in size. Implementation of larger values is encouraged whenever possible.

7. Proxy to SNMPv1

Historically, in order to support proxy to SNMPv1, as defined in [RFC2576], it was deemed useful to define a transport domain, rfc1157Domain, which indicates the transport mapping for SNMP messages as defined in [RFC1157].

8. Serialization using the Basic Encoding Rules

When the Basic Encoding Rules [BER] are used for serialization:

(1) When encoding the length field, only the definite form is used; use of the indefinite form encoding is prohibited. Note that when using the definite-long form, it is permissible to use more than the minimum number of length octets necessary to encode the length field.

(2) When encoding the value field, the primitive form shall be used for all simple types, i.e., INTEGER, OCTET STRING, and OBJECT IDENTIFIER (either IMPLICIT or explicit). The constructed form of encoding shall be used only for structured types, i.e., a SEQUENCE or an IMPLICIT SEQUENCE.

(3) When encoding an object whose syntax is described using the BITS construct, the value is encoded as an OCTET STRING, in which all the named bits in (the definition of) the bitstring, commencing with the first bit and proceeding to the last bit, are placed in bits 8 (high order bit) to 1 (low order bit) of the first octet, followed by bits 8 to 1 of each subsequent octet in turn, followed by as many bits as are needed of the final subsequent octet, commencing with bit 8. Remaining bits, if any, of the final octet are set to zero on generation and ignored on receipt.

These restrictions apply to all aspects of ASN.1 encoding, including the message wrappers, protocol data units, and the data objects they contain.

8.1. Usage Example

As an example of applying the Basic Encoding Rules, suppose one wanted to encode an instance of the GetBulkRequest-PDU [RFC3416]:

```
  [5] IMPLICIT SEQUENCE {
          request-id      1414684022,
          non-repeaters   1,
          max-repetitions 2,
          variable-bindings {
              { name sysUpTime,
                value { unSpecified NULL } },
              { name ipNetToMediaPhysAddress,
                value { unSpecified NULL } },
              { name ipNetToMediaType,
                value { unSpecified NULL } }
          }
      }
```

Applying the BER, this may be encoded (in hexadecimal) as:

```
[5] IMPLICIT SEQUENCE          a5 82 00 39
    INTEGER                    02 04 54 52 5d 76
    INTEGER                    02 01 01
    INTEGER                    02 01 02
    SEQUENCE (OF)              30 2b
        SEQUENCE               30 0b
            OBJECT IDENTIFIER  06 07 2b 06 01 02 01 01 03
            NULL               05 00
        SEQUENCE               30 0d
            OBJECT IDENTIFIER  06 09 2b 06 01 02 01 04 16 01 02
            NULL               05 00
        SEQUENCE               30 0d
            OBJECT IDENTIFIER  06 09 2b 06 01 02 01 04 16 01 04
            NULL               05 00
```

Note that the initial SEQUENCE in this example was not encoded using the minimum number of length octets. (The first octet of the length, 82, indicates that the length of the content is encoded in the next two octets.)

9. Notice on Intellectual Property

10. Acknowledgments

This document is the product of the SNMPv3 Working Group. Some special thanks are in order to the following Working Group members:

Randy Bush
Jeffrey D. Case
Mike Daniele
Rob Frye
Lauren Heintz
Keith McCloghrie
Russ Mundy
David T. Perkins
Randy Presuhn
Aleksey Romanov
Juergen Schoenwaelder
Bert Wijnen

This version of the document, edited by Randy Presuhn, was initially based on the work of a design team whose members were:

Jeffrey D. Case
Keith McCloghrie
David T. Perkins
Randy Presuhn
Juergen Schoenwaelder

The previous versions of this document, edited by Keith McCloghrie, was the result of significant work by four major contributors:

Jeffrey D. Case
Keith McCloghrie
Marshall T. Rose
Steven Waldbusser

Additionally, the contributions of the SNMPv2 Working Group to the previous versions are also acknowledged. In particular, a special thanks is extended for the contributions of:

Alexander I. Alten
Dave Arneson
Uri Blumenthal
Doug Book
Kim Curran
Jim Galvin
Maria Greene
Iain Hanson
Dave Harrington
Nguyen Hien
Jeff Johnson
Michael Kornegay
Deirdre Kostick
David Levi
Daniel Mahoney
Bob Natale
Brian O'Keefe
Andrew Pearson
Dave Perkins
Randy Presuhn
Aleksey Romanov
Shawn Routhier
Jon Saperia
Juergen Schoenwaelder
Bob Stewart
Kaj Tesink
Glenn Waters
Bert Wijnen

11. IANA Considerations

The SNMPv2-TM MIB module requires the allocation of a single object identifier for its MODULE-IDENTITY. IANA has allocated this object identifier in the snmpModules subtree, defined in the SNMPv2-SMI MIB module.

12. Security Considerations

SNMPv1 by itself is not a secure environment. Even if the network itself is secure (for example by using IPSec), even then, there is no control as to who on the secure network is allowed to access and GET/SET (read/change) the objects accessible through a command responder application.

It is recommended that the implementors consider the security features as provided by the SNMPv3 framework. Specifically, the use of the User-based Security Model STD 62, RFC 3414 [RFC3414] and the View-based Access Control Model STD 62, RFC 3415 [RFC3415] is recommended.

It is then a customer/user responsibility to ensure that the SNMP entity giving access to a MIB is properly configured to give access to the objects only to those principals (users) that have legitimate rights to indeed GET or SET (change) them.

13. References

13.1. Normative References

[BER] Information processing systems - Open Systems Interconnection - Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1), International Organization for Standardization. International Standard 8825, December 1987.

[IS8072] Information processing systems - Open Systems Interconnection - Transport Service Definition, International Organization for Standardization. International Standard 8072, June 1986.

[IS8072A] Information processing systems - Open Systems Interconnection - Transport Service Definition - Addendum 1: Connectionless-mode Transmission, International Organization for Standardization. International Standard 8072/AD 1, December 1986.

[RFC768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980.

[RFC791] Postel, J., "Internet Protocol", STD 5, RFC 791, September 1981.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2578] McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M. and S. Waldbusser, "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, April 1999.

[RFC2579] McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M. and S. Waldbusser, "Textual Conventions for SMIv2", STD 58, RFC 2579, April 1999.

[RFC2580] McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M. and S. Waldbusser, "Conformance Statements for SMIv2", STD 58, RFC 2580, April 1999.

[RFC3414] Blumenthal, U. and B. Wijnen, "The User-Based Security Model (USM) for Version 3 of the Simple Network Management Protocol (SNMPv3)", STD 62, RFC 3414, December 2002.

[RFC3415] 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.

[RFC3416] Presuhn, R., Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3416, December 2002.

13.2. Informative References

[APPLETALK] Sidhu, G., Andrews, R. and A. Oppenheimer, Inside AppleTalk (second edition). Addison-Wesley, 1990.

[NOVELL] Network System Technical Interface Overview. Novell, Inc., June 1989.

[RFC1157] Case, J., Fedor, M., Schoffstall, M. and J. Davin, "Simple Network Management Protocol", STD 15, RFC 1157, May 1990.

[RFC1742] Waldbusser, S. and K. Frisa, "AppleTalk Management Information Base II", RFC 1742, January 1995.

[RFC2576] Frye, R., Levi, D., Routhier, S. and B. Wijnen, "Coexistence between Version 1, Version 2, and Version 3 of the Internet-Standard Network Management Framework", RFC 2576, March 2000.

[RFC3410] Case, J., Mundy, R., Partain, D. and B. Stewart, "Introduction and Applicability Statements for Internet-Standard Management Framework", RFC 3410, December 2002.

[RFC3419] Daniele, M. and J. Schoenwaelder, "Textual Conventions for Transport Addresses", RFC 3419, November 2002.

14. Changes from RFC 1906

This document differs from RFC 1906 only in editorial improvements. The protocol is unchanged.

15. Editor's Address

Randy Presuhn
BMC Software, Inc.
2141 North First Street
San Jose, CA 95131
USA

Phone: +1 408 546-1006
EMail: randy_presuhn@bmc.com

16. Full Copyright Statement

Acknowledgement

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

Network Working Group
Request for Comments: 3418
STD: 62
Obsoletes: 1907
Category: Standards Track

Editor of this version:
R. Presuhn
BMC Software, Inc.
Authors of previous version:
J. Case
SNMP Research, Inc.
K. McCloghrie
Cisco Systems, Inc.
M. Rose
Dover Beach Consulting, Inc.
S. Waldbusser
International Network Services

December 2002

Management Information Base (MIB) for the Simple Network Management Protocol (SNMP)

Status of this Memo

Copyright Notice

Abstract

This document defines managed objects which describe the behavior of a Simple Network Management Protocol (SNMP) entity. This document obsoletes RFC 1907, Management Information Base for Version 2 of the Simple Network Management Protocol (SNMPv2).

Table of Contents

1. The Internet-Standard Management Framework

For a detailed overview of the documents that describe the current Internet-Standard Management Framework, please refer to section 7 of RFC 3410 [RFC3410].

Managed objects are accessed via a virtual information store, termed the Management Information Base or MIB. MIB objects are generally accessed through the Simple Network Management Protocol (SNMP).

Objects in the MIB are defined using the mechanisms defined in the Structure of Management Information (SMI). This memo specifies a MIB module that is compliant to the SMIv2, which is described in STD 58, RFC 2578 [RFC2578], STD 58, RFC 2579 [RFC2579] and STD 58, RFC 2580 [RFC2580].

It is the purpose of this document to define managed objects which describe the behavior of an SNMP entity, as defined in the SNMP architecture STD 62, [RFC3411].

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, RFC 2119 [RFC2119].

2. Definitions

```
SNMPv2-MIB DEFINITIONS ::= BEGIN

IMPORTS
    MODULE-IDENTITY, OBJECT-TYPE, NOTIFICATION-TYPE,
    TimeTicks, Counter32, snmpModules, mib-2
        FROM SNMPv2-SMI
    DisplayString, TestAndIncr, TimeStamp
```

```
          FROM SNMPv2-TC
      MODULE-COMPLIANCE, OBJECT-GROUP, NOTIFICATION-GROUP
          FROM SNMPv2-CONF;

   snmpMIB MODULE-IDENTITY
       LAST-UPDATED "200210160000Z"
       ORGANIZATION "IETF SNMPv3 Working Group"
       CONTACT-INFO
               "WG-EMail:   snmpv3@lists.tislabs.com
                Subscribe:  snmpv3-request@lists.tislabs.com

                Co-Chair:   Russ Mundy
                            Network Associates Laboratories
                postal:     15204 Omega Drive, Suite 300
                            Rockville, MD 20850-4601
                            USA
                EMail:      mundy@tislabs.com
                phone:      +1 301 947-7107

                Co-Chair:   David Harrington
                            Enterasys Networks
                postal:     35 Industrial Way
                            P. O. Box 5005
                            Rochester, NH 03866-5005
                            USA
                EMail:      dbh@enterasys.com
                phone:      +1 603 337-2614

                Editor:     Randy Presuhn
                            BMC Software, Inc.
                postal:     2141 North First Street
                            San Jose, CA 95131
                            USA
                EMail:      randy_presuhn@bmc.com
                phone:      +1 408 546-1006"
       DESCRIPTION
               "The MIB module for SNMP entities.

                Copyright (C) The Internet Society (2002). This
                version of this MIB module is part of RFC 3418;
                see the RFC itself for full legal notices.
               "
       REVISION      "200210160000Z"
       DESCRIPTION
               "This revision of this MIB module was published as
                RFC 3418."
       REVISION      "199511090000Z"
       DESCRIPTION
```

```
                "This revision of this MIB module was published as
                 RFC 1907."
        REVISION      "199304010000Z"
        DESCRIPTION
                "The initial revision of this MIB module was published
                as RFC 1450."
        ::= { snmpModules 1 }

    snmpMIBObjects OBJECT IDENTIFIER ::= { snmpMIB 1 }

    --  ::= { snmpMIBObjects 1 }        this OID is obsolete
    --  ::= { snmpMIBObjects 2 }        this OID is obsolete
    --  ::= { snmpMIBObjects 3 }        this OID is obsolete

    -- the System group
    --
    -- a collection of objects common to all managed systems.

    system   OBJECT IDENTIFIER ::= { mib-2 1 }

    sysDescr OBJECT-TYPE
        SYNTAX      DisplayString (SIZE (0..255))
        MAX-ACCESS  read-only
        STATUS      current
        DESCRIPTION
                "A textual description of the entity.  This value should
                include the full name and version identification of
                the system's hardware type, software operating-system,
                and networking software."
        ::= { system 1 }

    sysObjectID OBJECT-TYPE
        SYNTAX      OBJECT IDENTIFIER
        MAX-ACCESS  read-only
        STATUS      current
        DESCRIPTION
                "The vendor's authoritative identification of the
                network management subsystem contained in the entity.
                This value is allocated within the SMI enterprises
                subtree (1.3.6.1.4.1) and provides an easy and
                unambiguous means for determining 'what kind of box' is
                being managed.  For example, if vendor 'Flintstones,
                Inc.' was assigned the subtree 1.3.6.1.4.1.424242,
                it could assign the identifier 1.3.6.1.4.1.424242.1.1
                to its 'Fred Router'."
        ::= { system 2 }

    sysUpTime OBJECT-TYPE
```

```
        SYNTAX      TimeTicks
        MAX-ACCESS  read-only
        STATUS      current
        DESCRIPTION
                "The time (in hundredths of a second) since the
                network management portion of the system was last
                re-initialized."
        ::= { system 3 }

    sysContact OBJECT-TYPE
        SYNTAX      DisplayString (SIZE (0..255))
        MAX-ACCESS  read-write
        STATUS      current
        DESCRIPTION
                "The textual identification of the contact person for
                this managed node, together with information on how
                to contact this person.  If no contact information is
                known, the value is the zero-length string."
        ::= { system 4 }

    sysName OBJECT-TYPE
        SYNTAX      DisplayString (SIZE (0..255))
        MAX-ACCESS  read-write
        STATUS      current
        DESCRIPTION
                "An administratively-assigned name for this managed
                node.  By convention, this is the node's fully-qualified
                domain name.  If the name is unknown, the value is
                the zero-length string."
        ::= { system 5 }

    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 }

    sysServices OBJECT-TYPE
        SYNTAX      INTEGER (0..127)
        MAX-ACCESS  read-only
        STATUS      current
        DESCRIPTION
                "A value which indicates the set of services that this
                entity may potentially offer.  The value is a sum.
```

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

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

                For systems including OSI protocols, layers 5 and 6
                may also be counted."
        ::= { system 7 }

    -- object resource information
    --
    -- a collection of objects which describe the SNMP entity's
    -- (statically and dynamically configurable) support of
    -- various MIB modules.

    sysORLastChange OBJECT-TYPE
        SYNTAX     TimeStamp
        MAX-ACCESS read-only
        STATUS     current
        DESCRIPTION
                "The value of sysUpTime at the time of the most recent
                change in state or value of any instance of sysORID."
        ::= { system 8 }

    sysORTable OBJECT-TYPE
        SYNTAX     SEQUENCE OF SysOREntry
        MAX-ACCESS not-accessible
        STATUS     current
        DESCRIPTION
                "The (conceptual) table listing the capabilities of
                the local SNMP application acting as a command
                responder with respect to various MIB modules.
                SNMP entities having dynamically-configurable support
                of MIB modules will have a dynamically-varying number
                of conceptual rows."
        ::= { system 9 }
```

```
   sysOREntry OBJECT-TYPE
       SYNTAX     SysOREntry
       MAX-ACCESS not-accessible
       STATUS     current
       DESCRIPTION
               "An entry (conceptual row) in the sysORTable."
       INDEX      { sysORIndex }
       ::= { sysORTable 1 }

   SysOREntry ::= SEQUENCE {
       sysORIndex     INTEGER,
       sysORID        OBJECT IDENTIFIER,
       sysORDescr     DisplayString,
       sysORUpTime    TimeStamp
   }

   sysORIndex OBJECT-TYPE
       SYNTAX     INTEGER (1..2147483647)
       MAX-ACCESS not-accessible
       STATUS     current
       DESCRIPTION
               "The auxiliary variable used for identifying instances
               of the columnar objects in the sysORTable."
       ::= { sysOREntry 1 }

   sysORID OBJECT-TYPE
       SYNTAX     OBJECT IDENTIFIER
       MAX-ACCESS read-only
       STATUS     current
       DESCRIPTION
               "An authoritative identification of a capabilities
               statement with respect to various MIB modules supported
               by the local SNMP application acting as a command
               responder."
       ::= { sysOREntry 2 }

   sysORDescr OBJECT-TYPE
       SYNTAX     DisplayString
       MAX-ACCESS read-only
       STATUS     current
       DESCRIPTION
               "A textual description of the capabilities identified
               by the corresponding instance of sysORID."
       ::= { sysOREntry 3 }

   sysORUpTime OBJECT-TYPE
       SYNTAX     TimeStamp
       MAX-ACCESS read-only
```

```
        STATUS     current
        DESCRIPTION
                "The value of sysUpTime at the time this conceptual
                row was last instantiated."
        ::= { sysOREntry 4 }


    -- the SNMP group
    --
    -- a collection of objects providing basic instrumentation and
    -- control of an SNMP entity.

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

    snmpInPkts OBJECT-TYPE
        SYNTAX     Counter32
        MAX-ACCESS read-only
        STATUS     current
        DESCRIPTION
                "The total number of messages delivered to the SNMP
                entity from the transport service."
        ::= { snmp 1 }

    snmpInBadVersions OBJECT-TYPE
        SYNTAX     Counter32
        MAX-ACCESS read-only
        STATUS     current
        DESCRIPTION
                "The total number of SNMP messages which were delivered
                to the SNMP entity and were for an unsupported SNMP
                version."
        ::= { snmp 3 }

    snmpInBadCommunityNames OBJECT-TYPE
        SYNTAX     Counter32
        MAX-ACCESS read-only
        STATUS     current
        DESCRIPTION
               "The total number of community-based SNMP messages (for
               example,  SNMPv1) delivered to the SNMP entity which
               used an SNMP community name not known to said entity.
               Also, implementations which authenticate community-based
               SNMP messages using check(s) in addition to matching
               the community name (for example, by also checking
               whether the message originated from a transport address
               allowed to use a specified community name) MAY include
               in this value the number of messages which failed the
               additional check(s).  It is strongly RECOMMENDED that
```

```
              the documentation for any security model which is used
              to authenticate community-based SNMP messages specify
              the precise conditions that contribute to this value."
       ::= { snmp 4 }

   snmpInBadCommunityUses OBJECT-TYPE
       SYNTAX     Counter32
       MAX-ACCESS read-only
       STATUS     current
       DESCRIPTION
              "The total number of community-based SNMP messages (for
              example, SNMPv1) delivered to the SNMP entity which
              represented an SNMP operation that was not allowed for
              the SNMP community named in the message.  The precise
              conditions under which this counter is incremented
              (if at all) depend on how the SNMP entity implements
              its access control mechanism and how its applications
              interact with that access control mechanism.  It is
              strongly RECOMMENDED that the documentation for any
              access control mechanism which is used to control access
              to and visibility of MIB instrumentation specify the
              precise conditions that contribute to this value."
       ::= { snmp 5 }

   snmpInASNParseErrs OBJECT-TYPE
       SYNTAX     Counter32
       MAX-ACCESS read-only
       STATUS     current
       DESCRIPTION
               "The total number of ASN.1 or BER errors encountered by
               the SNMP entity when decoding received SNMP messages."
       ::= { snmp 6 }

   snmpEnableAuthenTraps OBJECT-TYPE
       SYNTAX      INTEGER { enabled(1), disabled(2) }
       MAX-ACCESS  read-write
       STATUS      current
       DESCRIPTION
               "Indicates whether the SNMP entity is permitted to
               generate authenticationFailure traps.  The value of this
               object overrides any configuration information; as such,
               it provides a means whereby all authenticationFailure
               traps may be disabled.

               Note that it is strongly recommended that this object
               be stored in non-volatile memory so that it remains
               constant across re-initializations of the network
               management system."
```

```
        ::= { snmp 30 }

    snmpSilentDrops OBJECT-TYPE
        SYNTAX     Counter32
        MAX-ACCESS read-only
        STATUS     current
        DESCRIPTION
               "The total number of Confirmed Class PDUs (such as
               GetRequest-PDUs, GetNextRequest-PDUs,
               GetBulkRequest-PDUs, SetRequest-PDUs, and
               InformRequest-PDUs) delivered to the SNMP entity which
               were silently dropped because the size of a reply
               containing an alternate Response Class PDU (such as a
               Response-PDU) with an empty variable-bindings field
               was greater than either a local constraint or the
               maximum message size associated with the originator of
               the request."
        ::= { snmp 31 }

    snmpProxyDrops OBJECT-TYPE
        SYNTAX     Counter32
        MAX-ACCESS read-only
        STATUS     current
        DESCRIPTION
                "The total number of Confirmed Class PDUs
                (such as GetRequest-PDUs, GetNextRequest-PDUs,
                GetBulkRequest-PDUs, SetRequest-PDUs, and
                InformRequest-PDUs) delivered to the SNMP entity which
                were silently dropped because the transmission of
                the (possibly translated) message to a proxy target
                failed in a manner (other than a time-out) such that
                no Response Class PDU (such as a Response-PDU) could
                be returned."
        ::= { snmp 32 }

    -- information for notifications
    --
    -- a collection of objects which allow the SNMP entity, when
    -- supporting a notification originator application,
    -- to be configured to generate SNMPv2-Trap-PDUs.

    snmpTrap       OBJECT IDENTIFIER ::= { snmpMIBObjects 4 }

    snmpTrapOID OBJECT-TYPE
        SYNTAX     OBJECT IDENTIFIER
        MAX-ACCESS accessible-for-notify
        STATUS     current
        DESCRIPTION
```

```
                "The authoritative identification of the notification
                currently being sent.  This variable occurs as
                the second varbind in every SNMPv2-Trap-PDU and
                InformRequest-PDU."
        ::= { snmpTrap 1 }

    --  ::= { snmpTrap 2 }   this OID is obsolete

    snmpTrapEnterprise OBJECT-TYPE
        SYNTAX     OBJECT IDENTIFIER
        MAX-ACCESS accessible-for-notify
        STATUS     current
        DESCRIPTION
                "The authoritative identification of the enterprise
                associated with the trap currently being sent.  When an
                SNMP proxy agent is mapping an RFC1157 Trap-PDU
                into a SNMPv2-Trap-PDU, this variable occurs as the
                last varbind."
        ::= { snmpTrap 3 }

    --  ::= { snmpTrap 4 }   this OID is obsolete


    -- well-known traps

    snmpTraps      OBJECT IDENTIFIER ::= { snmpMIBObjects 5 }

    coldStart NOTIFICATION-TYPE
        STATUS  current
        DESCRIPTION
                "A coldStart trap signifies that the SNMP entity,
                supporting a notification originator application, is
                reinitializing itself and that its configuration may
                have been altered."
        ::= { snmpTraps 1 }

    warmStart NOTIFICATION-TYPE
        STATUS  current
        DESCRIPTION
                "A warmStart trap signifies that the SNMP entity,
                supporting a notification originator application,
                is reinitializing itself such that its configuration
                is unaltered."
        ::= { snmpTraps 2 }

    -- Note the linkDown NOTIFICATION-TYPE ::= { snmpTraps 3 }
    -- and the linkUp NOTIFICATION-TYPE ::= { snmpTraps 4 }
    -- are defined in RFC 2863 [RFC2863]
```

```
   authenticationFailure NOTIFICATION-TYPE
       STATUS  current
       DESCRIPTION
               "An authenticationFailure trap signifies that the SNMP
                entity has received a protocol message that is not
                properly authenticated.  While all implementations
                of SNMP entities MAY be capable of generating this
                trap, the snmpEnableAuthenTraps object indicates
                whether this trap will be generated."
       ::= { snmpTraps 5 }

   -- Note the egpNeighborLoss notification is defined
   -- as { snmpTraps 6 } in RFC 1213

   -- the set group
   --
   -- a collection of objects which allow several cooperating
   -- command generator applications to coordinate their use of the
   -- set operation.

   snmpSet        OBJECT IDENTIFIER ::= { snmpMIBObjects 6 }

   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 }

   -- conformance information

   snmpMIBConformance
                  OBJECT IDENTIFIER ::= { snmpMIB 2 }

   snmpMIBCompliances
                  OBJECT IDENTIFIER ::= { snmpMIBConformance 1 }
   snmpMIBGroups  OBJECT IDENTIFIER ::= { snmpMIBConformance 2 }

   -- compliance statements
```

```
   --    ::= { snmpMIBCompliances 1 }      this OID is obsolete
   snmpBasicCompliance MODULE-COMPLIANCE
       STATUS  deprecated
       DESCRIPTION
               "The compliance statement for SNMPv2 entities which
               implement the SNMPv2 MIB.

               This compliance statement is replaced by
               snmpBasicComplianceRev2."
       MODULE  -- this module
           MANDATORY-GROUPS { snmpGroup, snmpSetGroup, systemGroup,
                              snmpBasicNotificationsGroup }

           GROUP   snmpCommunityGroup
           DESCRIPTION
               "This group is mandatory for SNMPv2 entities which
               support community-based authentication."

       ::= { snmpMIBCompliances 2 }

   snmpBasicComplianceRev2 MODULE-COMPLIANCE
       STATUS  current
       DESCRIPTION
               "The compliance statement for SNMP entities which
               implement this MIB module."
       MODULE  -- this module
           MANDATORY-GROUPS { snmpGroup, snmpSetGroup, systemGroup,
                              snmpBasicNotificationsGroup }

           GROUP   snmpCommunityGroup
           DESCRIPTION
               "This group is mandatory for SNMP entities which
               support community-based authentication."

           GROUP   snmpWarmStartNotificationGroup
           DESCRIPTION
               "This group is mandatory for an SNMP entity which
               supports command responder applications, and is
               able to reinitialize itself such that its
               configuration is unaltered."

       ::= { snmpMIBCompliances 3 }

   -- units of conformance

   --  ::= { snmpMIBGroups 1 }           this OID is obsolete
   --  ::= { snmpMIBGroups 2 }           this OID is obsolete
   --  ::= { snmpMIBGroups 3 }           this OID is obsolete
```

```
   --  ::= { snmpMIBGroups 4 }           this OID is obsolete

   snmpGroup OBJECT-GROUP
       OBJECTS { snmpInPkts,
                 snmpInBadVersions,
                 snmpInASNParseErrs,
                 snmpSilentDrops,
                 snmpProxyDrops,
                 snmpEnableAuthenTraps }
       STATUS  current
       DESCRIPTION
               "A collection of objects providing basic instrumentation
               and control of an SNMP entity."
       ::= { snmpMIBGroups 8 }

   snmpCommunityGroup OBJECT-GROUP
       OBJECTS { snmpInBadCommunityNames,
                 snmpInBadCommunityUses }
       STATUS  current
       DESCRIPTION
               "A collection of objects providing basic instrumentation
               of a SNMP entity which supports community-based
               authentication."
       ::= { snmpMIBGroups 9 }

   snmpSetGroup OBJECT-GROUP
       OBJECTS { snmpSetSerialNo }
       STATUS  current
       DESCRIPTION
               "A collection of objects which allow several cooperating
               command generator applications to coordinate their
               use of the set operation."
       ::= { snmpMIBGroups 5 }

   systemGroup OBJECT-GROUP
       OBJECTS { sysDescr, sysObjectID, sysUpTime,
                 sysContact, sysName, sysLocation,
                 sysServices,
                 sysORLastChange, sysORID,
                 sysORUpTime, sysORDescr }
       STATUS  current
       DESCRIPTION
               "The system group defines objects which are common to all
               managed systems."
       ::= { snmpMIBGroups 6 }

   snmpBasicNotificationsGroup NOTIFICATION-GROUP
       NOTIFICATIONS { coldStart, authenticationFailure }
```

```
        STATUS       current
        DESCRIPTION
           "The basic notifications implemented by an SNMP entity
            supporting command responder applications."
        ::= { snmpMIBGroups 7 }

    snmpWarmStartNotificationGroup NOTIFICATION-GROUP
       NOTIFICATIONS { warmStart }
       STATUS       current
       DESCRIPTION
         "An additional notification for an SNMP entity supporting
         command responder applications, if it is able to reinitialize
         itself such that its configuration is unaltered."
      ::= { snmpMIBGroups 11 }

    snmpNotificationGroup OBJECT-GROUP
        OBJECTS { snmpTrapOID, snmpTrapEnterprise }
        STATUS  current
        DESCRIPTION
                "These objects are required for entities
                which support notification originator applications."
        ::= { snmpMIBGroups 12 }

    -- definitions in RFC 1213 made obsolete by the inclusion of a
    -- subset of the snmp group in this MIB

    snmpOutPkts OBJECT-TYPE
        SYNTAX      Counter32
        MAX-ACCESS  read-only
        STATUS      obsolete
        DESCRIPTION
                "The total number of SNMP Messages which were
                passed from the SNMP protocol entity to the
                transport service."
        ::= { snmp 2 }

    -- { snmp 7 } is not used

    snmpInTooBigs OBJECT-TYPE
        SYNTAX      Counter32
        MAX-ACCESS  read-only
        STATUS      obsolete
        DESCRIPTION
                "The total number of SNMP PDUs which were
                delivered to the SNMP protocol entity and for
                which the value of the error-status field was
                'tooBig'."
        ::= { snmp 8 }
```

```
   snmpInNoSuchNames OBJECT-TYPE
       SYNTAX      Counter32
       MAX-ACCESS  read-only
       STATUS      obsolete
       DESCRIPTION
               "The total number of SNMP PDUs which were
               delivered to the SNMP protocol entity and for
               which the value of the error-status field was
               'noSuchName'."
       ::= { snmp 9 }

   snmpInBadValues OBJECT-TYPE
       SYNTAX      Counter32
       MAX-ACCESS  read-only
       STATUS      obsolete
       DESCRIPTION
               "The total number of SNMP PDUs which were
               delivered to the SNMP protocol entity and for
               which the value of the error-status field was
               'badValue'."
       ::= { snmp 10 }

   snmpInReadOnlys OBJECT-TYPE
       SYNTAX      Counter32
       MAX-ACCESS  read-only
       STATUS      obsolete
       DESCRIPTION
               "The total number valid SNMP PDUs which were delivered
               to the SNMP protocol entity and for which the value
               of the error-status field was 'readOnly'.  It should
               be noted that it is a protocol error to generate an
               SNMP PDU which contains the value 'readOnly' in the
               error-status field, as such this object is provided
               as a means of detecting incorrect implementations of
               the SNMP."
       ::= { snmp 11 }

   snmpInGenErrs OBJECT-TYPE
       SYNTAX      Counter32
       MAX-ACCESS  read-only
       STATUS      obsolete
       DESCRIPTION
               "The total number of SNMP PDUs which were delivered
               to the SNMP protocol entity and for which the value
               of the error-status field was 'genErr'."
       ::= { snmp 12 }

   snmpInTotalReqVars OBJECT-TYPE
```

```
        SYNTAX      Counter32
        MAX-ACCESS  read-only
        STATUS      obsolete
        DESCRIPTION
                "The total number of MIB objects which have been
                retrieved successfully by the SNMP protocol entity
                as the result of receiving valid SNMP Get-Request
                and Get-Next PDUs."
        ::= { snmp 13 }

    snmpInTotalSetVars OBJECT-TYPE
        SYNTAX      Counter32
        MAX-ACCESS  read-only
        STATUS      obsolete
        DESCRIPTION
                "The total number of MIB objects which have been
                altered successfully by the SNMP protocol entity as
                the result of receiving valid SNMP Set-Request PDUs."
        ::= { snmp 14 }

    snmpInGetRequests OBJECT-TYPE
        SYNTAX      Counter32
        MAX-ACCESS  read-only
        STATUS      obsolete
        DESCRIPTION
                "The total number of SNMP Get-Request PDUs which
                have been accepted and processed by the SNMP
                protocol entity."
        ::= { snmp 15 }

    snmpInGetNexts OBJECT-TYPE
        SYNTAX      Counter32
        MAX-ACCESS  read-only
        STATUS      obsolete
        DESCRIPTION
                "The total number of SNMP Get-Next PDUs which have been
                accepted and processed by the SNMP protocol entity."
        ::= { snmp 16 }

    snmpInSetRequests OBJECT-TYPE
        SYNTAX      Counter32
        MAX-ACCESS  read-only
        STATUS      obsolete
        DESCRIPTION
                "The total number of SNMP Set-Request PDUs which
                have been accepted and processed by the SNMP protocol
                entity."
        ::= { snmp 17 }
```

```
   snmpInGetResponses OBJECT-TYPE
       SYNTAX      Counter32
       MAX-ACCESS  read-only
       STATUS      obsolete
       DESCRIPTION
               "The total number of SNMP Get-Response PDUs which
               have been accepted and processed by the SNMP protocol
               entity."
       ::= { snmp 18 }

   snmpInTraps OBJECT-TYPE
       SYNTAX      Counter32
       MAX-ACCESS  read-only
       STATUS      obsolete
       DESCRIPTION
               "The total number of SNMP Trap PDUs which have been
               accepted and processed by the SNMP protocol entity."
       ::= { snmp 19 }

   snmpOutTooBigs OBJECT-TYPE
       SYNTAX      Counter32
       MAX-ACCESS  read-only
       STATUS      obsolete
       DESCRIPTION
               "The total number of SNMP PDUs which were generated
               by the SNMP protocol entity and for which the value
               of the error-status field was 'tooBig.'"
       ::= { snmp 20 }

   snmpOutNoSuchNames OBJECT-TYPE
       SYNTAX      Counter32
       MAX-ACCESS  read-only
       STATUS      obsolete
       DESCRIPTION
               "The total number of SNMP PDUs which were generated
               by the SNMP protocol entity and for which the value
               of the error-status was 'noSuchName'."
       ::= { snmp 21 }

   snmpOutBadValues OBJECT-TYPE
       SYNTAX      Counter32
       MAX-ACCESS  read-only
       STATUS      obsolete
       DESCRIPTION
               "The total number of SNMP PDUs which were generated
               by the SNMP protocol entity and for which the value
               of the error-status field was 'badValue'."
       ::= { snmp 22 }
```

```
   -- { snmp 23 } is not used

   snmpOutGenErrs OBJECT-TYPE
       SYNTAX      Counter32
       MAX-ACCESS  read-only
       STATUS      obsolete
       DESCRIPTION
               "The total number of SNMP PDUs which were generated
               by the SNMP protocol entity and for which the value
               of the error-status field was 'genErr'."
       ::= { snmp 24 }

   snmpOutGetRequests OBJECT-TYPE
       SYNTAX      Counter32
       MAX-ACCESS  read-only
       STATUS      obsolete
       DESCRIPTION
               "The total number of SNMP Get-Request PDUs which
               have been generated by the SNMP protocol entity."
       ::= { snmp 25 }

   snmpOutGetNexts OBJECT-TYPE
       SYNTAX      Counter32
       MAX-ACCESS  read-only
       STATUS      obsolete
       DESCRIPTION
               "The total number of SNMP Get-Next PDUs which have
               been generated by the SNMP protocol entity."
       ::= { snmp 26 }

   snmpOutSetRequests OBJECT-TYPE
       SYNTAX      Counter32
       MAX-ACCESS  read-only
       STATUS      obsolete
       DESCRIPTION
               "The total number of SNMP Set-Request PDUs which
               have been generated by the SNMP protocol entity."
       ::= { snmp 27 }

   snmpOutGetResponses OBJECT-TYPE
       SYNTAX      Counter32
       MAX-ACCESS  read-only
       STATUS      obsolete
       DESCRIPTION
               "The total number of SNMP Get-Response PDUs which
               have been generated by the SNMP protocol entity."
       ::= { snmp 28 }
```

```
   snmpOutTraps OBJECT-TYPE
       SYNTAX      Counter32
       MAX-ACCESS  read-only
       STATUS      obsolete
       DESCRIPTION
               "The total number of SNMP Trap PDUs which have
               been generated by the SNMP protocol entity."
       ::= { snmp 29 }

   snmpObsoleteGroup OBJECT-GROUP
       OBJECTS { snmpOutPkts, snmpInTooBigs, snmpInNoSuchNames,
                 snmpInBadValues, snmpInReadOnlys, snmpInGenErrs,
                 snmpInTotalReqVars, snmpInTotalSetVars,
                 snmpInGetRequests, snmpInGetNexts, snmpInSetRequests,
                 snmpInGetResponses, snmpInTraps, snmpOutTooBigs,
                 snmpOutNoSuchNames, snmpOutBadValues,
                 snmpOutGenErrs, snmpOutGetRequests, snmpOutGetNexts,
                 snmpOutSetRequests, snmpOutGetResponses, snmpOutTraps
                 }
       STATUS  obsolete
       DESCRIPTION
               "A collection of objects from RFC 1213 made obsolete
               by this MIB module."
       ::= { snmpMIBGroups 10 }

   END
```

3. Notice on Intellectual Property

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-related documentation can be found in BCP-11. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to 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.

4. Acknowledgments

This document is the product of the SNMPv3 Working Group. Some special thanks are in order to the following Working Group members:

Randy Bush
Jeffrey D. Case
Mike Daniele
Rob Frye
Lauren Heintz
Keith McCloghrie
Russ Mundy
David T. Perkins
Randy Presuhn
Aleksey Romanov
Juergen Schoenwaelder
Bert Wijnen

This version of the document, edited by Randy Presuhn, was initially based on the work of a design team whose members were:

Jeffrey D. Case
Keith McCloghrie
David T. Perkins
Randy Presuhn
Juergen Schoenwaelder

The previous versions of this document, edited by Keith McCloghrie, was the result of significant work by four major contributors:

Jeffrey D. Case
Keith McCloghrie
Marshall T. Rose
Steven Waldbusser

Additionally, the contributions of the SNMPv2 Working Group to the previous versions are also acknowledged. In particular, a special thanks is extended for the contributions of:

Alexander I. Alten
Dave Arneson
Uri Blumenthal
Doug Book
Kim Curran
Jim Galvin
Maria Greene
Iain Hanson
Dave Harrington
Nguyen Hien
Jeff Johnson
Michael Kornegay
Deirdre Kostick
David Levi
Daniel Mahoney
Bob Natale
Brian O'Keefe
Andrew Pearson
Dave Perkins
Randy Presuhn
Aleksey Romanov
Shawn Routhier
Jon Saperia
Juergen Schoenwaelder
Bob Stewart
Kaj Tesink
Glenn Waters
Bert Wijnen

5. Security Considerations

There are a number of management objects defined in this MIB that have a MAX-ACCESS clause of read-write. Such objects may be considered sensitive or vulnerable in some network environments. The support for SET operations in a non-secure environment without proper protection can have a negative effect on network operations.

SNMPv1 by itself is not a secure environment. Even if the network itself is secure (for example by using IPSec), even then, there is no control as to who on the secure network is allowed to access and GET/SET (read/change) the objects in this MIB.

It is recommended that the implementors consider the security features as provided by the SNMPv3 framework. Specifically, the use of the User-based Security Model STD 62, RFC 3414 [RFC3414] and the View-based Access Control Model STD 62, RFC 3415 [RFC3415] is recommended.

It is then a customer/user responsibility to ensure that the SNMP entity giving access to an instance of this MIB is properly configured to give access to the objects only to those principals (users) that have legitimate rights to indeed GET or SET (change) them.

6. References

6.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2578] McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M. and S. Waldbusser, "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, April 1999.

[RFC2579] McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M. and S. Waldbusser, "Textual Conventions for SMIv2", STD 58, RFC 2579, April 1999.

[RFC2580] McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M. and S. Waldbusser, "Conformance Statements for SMIv2", STD 58, RFC 2580, April 1999.

[RFC3411] Harrington, D., Presuhn, R. and B. Wijnen, "An Architecture for describing Simple Network Management Protocol (SNMP) Management Frameworks", STD 62, RFC 3411, December 2002.

[RFC3414] Blumenthal, U. and B. Wijnen, "The User-Based Security Model (USM) for Version 3 of the Simple Network Management Protocol (SNMPv3)", STD 62, RFC 3414, December 2002.

[RFC3415] 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.

6.1. Informative References

[RFC1157] Case, J., Fedor, M., Schoffstall, M. and J. Davin, "Simple Network Management Protocol", STD 15, RFC 1157, May 1990.

[RFC1213] McCloghrie, K. and M. Rose, "Management Information Base for Network Management of TCP/IP-based internets: MIB-II", STD 16, RFC 1213, March 1991.

[RFC2863] McCloghrie, K. and F. Kastenholz, "The Interfaces Group MIB", RFC 2863, June 2000.

[RFC3410] Case, J., Mundy, R., Partain, D. and B. Stewart, "Introduction and Applicability Statements for Internet-Standard Management Framework", RFC 3410, December 2002.

7. Changes from RFC 1907

These are the changes from RFC 1907:

- Corrected typo in copyright statement;
- Updated copyright date;
- Updated with new editor’s name and contact information;
- Cosmetic fixes to layout and typography;
- Changed title;
- Replace introduction with current MIB boilerplate;
- Updated references;
- Fixed typo in sysORUpTime;
- Re-worded description of snmpSilentDrops;
- Updated reference to RFC 1573 to 2863;
- Added IPR boilerplate as required by RFC 2026;
- Weakened authenticationFailure description from MUST to MAY, clarified that it pertains to all SNMP entities;

- Clarified descriptions of snmpInBadCommunityNames and snmpInBadCommunityUses;

- Updated module-identity and contact information;

- Updated the acknowledgments section;

- Replaced references to "manager role", "agent role" and "SNMPv2 entity" with appropriate terms from RFC 2571;

- Updated document headers and footers;

- Added security considerations, based on current recommendations for MIB modules;

- Added NOTIFICATION-GROUP and OBJECT-GROUP constructs for NOTIFICATION-TYPEs and OBJECT-TYPEs that were left unreferenced in RFC 1907;

- Fixed typos in sysServices DESCRIPTION;

- Changed description of snmpProxyDrops to use terms from architecture;

- Changed value used in example for sysObjectID;

- Added an abstract;

- Deprecated the snmpBasicCompliance MODULE-COMPLIANCE, and added the snmpBasicComplianceRev2 MODULE-COMPLIANCE to take its place;

- Updated working group mailing list address;

- Added co-chair’s address.

8. Editor’s Address

Randy Presuhn
BMC Software, Inc.
2141 North First Street
San Jose, CA 95131
USA

Phone: +1 408 546 1006
EMail: randy_presuhn@bmc.com

9. Full Copyright Statement

Acknowledgement

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