ORDER

U.S. DEPARTMENT OF TRANSPORTATION
FEDERAL AVIATION ADMINISTRATION

6000.53

3/27/00

SUBJ: REMOTE MAINTENANCE MONITORING INTERFACES

1. PURPOSE. This order establishes the requirement that all National Airspace System (NAS) subsystems shall provide Remote Maintenance Monitoring (RMM) capability.

2. DISTRIBUTION. This order is distributed to the division level in Airway Facilities; Air Traffic; Air Traffic System Requirements; Flight Standards; Offices of Independent Operational Test and Evaluation; System Architecture and Investment Analysis; Communications, Navigation, and Surveillance Systems; Information Technology; Acquisitions; Air Traffic Systems Development; and Airport Safety and Standards in Washington; to the division level at the FAA Academy, the Logistics Center, and the Technical Center; to all Airway Facilities, Air Traffic, and Flight Standards regional offices; and to all Airway Facilities, Air Traffic, and Flight Standards field offices with a standard distribution.

3. BACKGROUND. The Federal Aviation Administration (FAA) will be required to meet an increasing demand for user services. In response to this requirement, the NAS is being extensively modernized with state-of-the-art, highly reliable equipment. A remote maintenance monitoring system (RMMS) is being implemented to facilitate efficient management of NAS services. The purpose of RMMS is to support the Operations Control Center (OCC) approach, enhance the integrity of the NAS, reduce maintenance and support costs, and minimize the impact of equipment failures. The monitoring system will improve personnel productivity through the use of centralized maintenance techniques.

4. REQUIREMENTS.

a. All future NAS subsystems shall provide RMM capability and shall be integrated into the National Airspace System Infrastructure Management System (NIMS). Existing NAS subsystems shall be retrofitted with RMM capability and shall be integrated into NIMS to the extent determined to be cost effective. RMM interfaces for subsystems having benefit-to-cost ratios less than one shall not be required unless there are overriding operational requirements that cannot be satisfied by means other than RMM.

b. All NAS subsystems shall use an open system interface; e.g., Simple NetworkManagement Protocol (SNMP) or Common Management Information Protocol (CMIP) to connect RMMS to NIMS. The following signed baseline documents will be used for guidance:

(1) NAS-IR-51070000, Interface Requirements Document, NAS Infrastructure Management System Manager/Managed Subsystem, 05/28/97.

(2) NAS-IC-51070000-1, Interface Control Document, NAS Infrastructure Management System Manager/Managed Subsystem using the Simple Network Management Protocol Version I (SNMPv1), 07/22/97.

c. All NAS programs shall coordinate with the NAS Operations Program (AOP) for RMM requirements. All decisions to exclude RMM on new NAS subsystems shall be coordinated with AOP and will require a waiver signed by the Director of Airway Facilities.

5. REFERENCE. The latest edition of Order 6000.30, National Airspace System (NAS) Maintenance Policy, identifies the RMM requirements for Airway Facilities.

Alan R. Moore

Director of Airway Facilities

Network Working Group
Request for Comments: 2570
Category: Informational

J. Case
SNMP Research, Inc.
R. Mundy
TIS Labs at Network Associates, Inc.
D. Partain
Ericsson
B. Stewart
Cisco Systems
April 1999

Introduction to Version 3 of the Internet-standard Network Management Framework

Status of this Memo

This memo provides information for the Internet community. It does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

Copyright Notice

Copyright (C) The Internet Society (1999). All Rights Reserved.

Abstract

The purpose of this document is to provide an overview of the third version of the Internet-standard Management Framework, termed the SNMP version 3 Framework (SNMPv3). This Framework is derived from and builds upon both the original Internet-standard Management Framework (SNMPv1) and the second Internet-standard Management Framework (SNMPv2).

The architecture is designed to be modular to allow the evolution of the Framework over time.

Table of Contents

1 Introduction 2
2 The Internet Standard Management Framework 3
2.1 Basic Structure and Components 3
2.2 Architecture of the Internet Standard Management Framework 3
3 The SNMPv1 Management Framework 4
3.1 The SNMPv1 Data Definition Language 5
3.2 Management Information 6
3.3 Protocol Operations 6
3.4 SNMPv1 Security and Administration 6

4 The SNMPv2 Management Framework 7
5 The SNMPv3 Working Group 8
6 SNMPv3 Framework Module Specifications 10
6.1 Data Definition Language 10
6.2 MIB Modules 11
6.3 Protocol Operations and Transport Mappings 12
6.4 SNMPv3 Security and Administration 12
7 Document Summaries 13
7.1 Structure of Management Information 13
7.1.1 Base SMI Specification 13
7.1.2 Textual Conventions 14
7.1.3 Conformance Statements 15
7.2 Protocol Operations 15
7.3 Transport Mappings 15
7.4 Protocol Instrumentation 16
7.5 Architecture / Security and Administration 16
7.6 Message Processing and Dispatch (MPD) 16
7.7 SNMP Applications 17
7.8 User-based Security Model (USM) 17
7.9 View-based Access Control (VACM) 18
7.10 SNMPv3 Coexistence and Transition 18
8 Security Considerations 19
9 Editors' Addresses 19
10 References 20
11 Full Copyright Statement 23

1 Introduction

This document is an introduction to the third version of the Internet-standard Management Framework, termed the SNMP version 3 Management Framework (SNMPv3) and has multiple purposes.

First, it describes the relationship between the SNMP version 3 (SNMPv3) specifications and the specifications of the SNMP version 1 (SNMPv1) Management Framework, the SNMP version 2 (SNMPv2) Management Framework, and the Community-based Administrative Framework for SNMPv2.

Second, it provides a roadmap to the multiple documents which contain the relevant specifications.

Third, this document provides a brief easy-to-read summary of the contents of each of the relevant specification documents.

This document is intentionally tutorial in nature and, as such, may occasionally be "guilty" of oversimplification. In the event of a conflict or contradiction between this document and the more detailed documents for which this document is a roadmap, the specifications in

the more detailed documents shall prevail.

Further, the detailed documents attempt to maintain separation between the various component modules in order to specify well-

defined interfaces between them. This roadmap document, however, takes a different approach and attempts to provide an integrated view of the various component modules in the interest of readability.

2 The Internet Standard Management Framework

The third version of the Internet Standard Management Framework (the SNMPv3 Framework) is derived from and builds upon both the original Internet-standard Management Framework (SNMPv1) and the second Internet-standard Management Framework (SNMPv2).

All versions (SNMPv1, SNMPv2, and SNMPv3) of the Internet Standard Management Framework share the same basic structure and components. Furthermore, all versions of the specifications of the Internet Standard Management Framework follow the same architecture.

2.1 Basic Structure and Components

An enterprise deploying the Internet Standard Management Framework contains four basic components:

* several (typically many) managed nodes, each with an SNMP entity which provides remote access to management instrumentation (traditionally called an agent);

* at least one SNMP entity with management applications (typically called a manager),

* a management protocol used to convey management information between the SNMP entities, and

* management information.

The management protocol is used to convey management information between SNMP entities such as managers and agents.

This basic structure is common to all versions of the Internet Standard Management Framework; i.e., SNMPv1, SNMPv2, and SNMPv3.

2.2 Architecture of the Internet Standard Management Framework

The specifications of the Internet Standard Management Framework are based on a modular architecture. This framework is more than just a protocol for moving data. It consists of:

* a data definition language,

* definitions of management information (the Management Information Base, or MIB),

* a protocol definition, and

* security and administration.

Over time, as the Framework has evolved from SNMPv1, through SNMPv2, to SNMPv3, the definitions of each of these architectural components have become richer and more clearly defined, but the fundamental architecture has remained consistent.

One prime motivator for this modularity was to enable the ongoing evolution of the Framework as is documented in RFC 1052 [14]. When originally envisioned, this capability was to be used to ease the transition from SNMP-based management of internets to management based on OSI protocols. To this end, the framework was architected with a protocol-independent data definition language and Management Information Base along with a MIB-independent protocol. This separation was designed to allow the SNMP-based protocol to be replaced without requiring the management information to be redefined or reinstrumented. History has shown that the selection of this architecture was the right decision for the wrong reason -- it turned out that this architecture has eased the transition from SNMPv1 to SNMPv2 and from SNMPv2 to SNMPv3 rather than easing the transition away from management based on the Simple Network Management Protocol.

The SNMPv3 Framework builds and extends these architectural principles by:

* building on these four basic architectural components, in some cases incorporating them from the SNMPv2 Framework by reference, and

* by using these same layering principles in the definition of new capabilities in the security and administration portion of the architecture.

Those who are familiar with the architecture of the SNMPv1 Management Framework and the SNMPv2 Management Framework will find many familiar concepts in the architecture of the SNMPv3 Management Framework. However, in some cases, the terminology may be somewhat different.

3 The SNMPv1 Management Framework

The original Internet-standard Network Management Framework (SNMPv1) is defined in the following documents:

* STD 16, RFC 1155 [1] which defines the Structure of Management Information (SMI), the mechanisms used for describing and naming objects for the purpose of management.

* STD 16, RFC 1212 [2] which defines a more concise description mechanism for describing and naming management information objects, but which is wholly consistent with the SMI.

* STD 15, RFC 1157 [3] which defines the Simple Network Management Protocol (SNMP), the protocol used for network access to managed objects and event notification. Note this document also defines an initial set of event notifications.

Additionally, two documents are generally considered to be companions to these three:

* STD 17, RFC 1213 [13] which contains definitions for the base set of management information

* RFC 1215 [25] defines a concise description mechanism for defining event notifications, which are called traps in the SNMPv1 protocol. It also specifies the generic traps from RFC 1157 in the concise notation.

These documents describe the four parts of the first version of the SNMP Framework.

3.1 The SNMPv1 Data Definition Language

The first two and the last document describe the SNMPv1 data definition language. Note that due to the initial requirement that the SMI be protocol-independent, the first two SMI documents do not provide a means for defining event notifications (traps). Instead, the SNMP protocol document defines a few standardized event notifications (generic traps) and provides a means for additional event notifications to be defined. The last document specifies a straight-forward approach towards defining event notifications used with the SNMPv1 protocol. At the time that it was written, use of traps in the Internet-standard network management framework was controversial. As such, RFC 1215 was put forward with the status of "Informational", which was never updated because it was believed that the second version of the SNMP Framework would replace the first version. Note that the SNMPv1 data definition language is sometimes

referred to as SMIv1.

3.2 Management Information

The data definition language described in the first two documents was first used to define the now-historic MIB-I as specified in RFC 1066 [12], and was subsequently used to define MIB-II as specified in RFC 1213 [13].

Later, after the publication of MIB-II, a different approach to management information definition was taken from the earlier approach of having a single committee staffed by generalists work on a single document to define the Internet-standard MIB. Rather, many mini-MIB documents were produced in a parallel and distributed fashion by groups chartered to produce a specification for a focused portion of the Internet-standard MIB and staffed by personnel with expertise in

those particular areas ranging from various aspects of network management, to system management, and application management.

3.3 Protocol Operations

The third document, STD 15, describes the SNMPv1 protocol operations performed by protocol data units (PDUs) on lists of variable bindings and describes the format of SNMPv1 messages. The operators defined by SNMPv1 are: get, get-next, get-response, set-request, and trap. Typical layering of SNMP on a connectionless transport service is also defined.

3.4 SNMPv1 Security and Administration

STD 15 also describes an approach to security and administration. Many of these concepts are carried forward and some, particularly security, are extended by the SNMPv3 Framework.

The SNMPv1 Framework describes the encapsulation of SNMPv1 PDUs in SNMP messages between SNMP entities and distinguishes between application entities and protocol entities. In SNMPv3, these are renamed applications and engines, respectively.

The SNMPv1 Framework also introduces the concept of an authentication service supporting one or more authentication schemes. In addition to authentication, SNMPv3 defines the additional security capability referred to as privacy. (Note: some literature from the security community would describe SNMPv3 security capabilities as providing data integrity, source authenticity, and confidentiality.) The modular nature of the SNMPv3 Framework permits both changes and additions to the security capabilities.

Finally, the SNMPv1 Framework introduces access control based on a concept called an SNMP MIB view. The SNMPv3 Framework specifies a fundamentally similar concept called view-based access control. With this capability, SNMPv3 provides the means for controlling access to information on managed devices.

However, while the SNMPv1 Framework anticipated the definition of multiple authentication schemes, it did not define any such schemes other than a trivial authentication scheme based on community strings. This was a known fundamental weakness in the SNMPv1 Framework but it was thought at that time that the definition of commercial grade security might be contentious in its design and difficult to get approved because "security" means many different things to different people. To that end, and because some users do not require strong authentication, the SNMPv1 architected an authentication service as a separate block to be defined "later" and the SNMPv3 Framework provides an architecture for use within that block as well as a definition for its subsystems.

4 The SNMPv2 Management Framework

The SNMPv2 Management Framework is fully described in [4-9] and coexistence and transition issues relating to SNMPv1 and SNMPv2 are discussed in [10].

SNMPv2 provides several advantages over SNMPv1, including:

* expanded data types (e.g., 64 bit counter)

* improved efficiency and performance (get-bulk operator)

* confirmed event notification (inform operator)

* richer error handling (errors and exceptions)

* improved sets, especially row creation and deletion

* fine tuning of the data definition language

However, the SNMPv2 Framework, as described in these documents, is incomplete in that it does not meet the original design goals of the SNMPv2 project. The unmet goals included provision of security and administration delivering so-called "commercial grade" security with

* authentication: origin identification, message integrity, and some aspects of replay protection;

* privacy: confidentiality;

* authorization and access control; and

* suitable remote configuration and administration capabilities for these features.

The SNMPv3 Management Framework, as described in this document and the companion documents, addresses these significant deficiencies.

5 The SNMPv3 Working Group

This document, and its companion documents, were produced by the SNMPv3 Working Group of the Internet Engineering Task Force (IETF). The SNMPv3 Working Group was chartered to prepare recommendations for the next generation of SNMP. The goal of the Working Group was to produce the necessary set of documents that provide a single standard for the next generation of core SNMP functions. The single, most critical need in the next generation is a definition of security and administration that makes SNMP-based management transactions secure in a way which is useful for users who wish to use SNMPv3 to manage networks, the systems that make up those networks, and the applications which reside on those systems, including manager-to-agent, agent-to-manager, and manager-to-manager transactions.

In the several years prior to the chartering of the Working Group,

there were a number of activities aimed at incorporating security and other improvements to SNMP. These efforts included:

* "SNMP Security" circa 1991-1992 [RFC 1351 - RFC 1353],

* "SMP" circa 1992-1993,

* "The Party-based SNMPv2" circa 1993-1995 [RFC 1441 - RFC 1452].

Each of these efforts incorporated commercial grade, industrial strength security including authentication, privacy, authorization, view-based access control, and administration, including remote configuration.

These efforts fed the development of the SNMPv2 Management Framework as described in RFCs 1902 - 1908. However, the Framework described in those RFCs had no standards-based security and administrative framework of its own; rather, it was associated with multiple security and administrative frameworks, including:

* "The Community-based SNMPv2" (SNMPv2c) [RFC 1901],

* "SNMPv2u" [RFCs 1909 - 1910] and

* "SNMPv2*".

SNMPv2c had the endorsement of the IETF but no security and administration whereas both SNMPv2u and SNMPv2* had security but lacked the endorsement of the IETF.

The SNMPv3 Working Group was chartered to produce a single set of specifications for the next generation of SNMP, based upon a convergence of the concepts and technical elements of SNMPv2u and SNMPv2*, as was suggested by an advisory team which was formed to provide a single recommended approach for SNMP evolution.

In so doing, the Working Group charter defined the following objectives:

* accommodate the wide range of operational environments with differing management demands;

* facilitate the need to transition from previous, multiple protocols to SNMPv3;

* facilitate the ease of setup and maintenance activities.

In the initial work of the SNMPv3 Working Group, the group focused on security and administration, including

* authentication and privacy,

* authorization and view-based access control, and

* standards-based remote configuration of the above.

The SNMPv3 Working Group did not "reinvent the wheel," but reused the SNMPv2 Draft Standard documents, i.e., RFCs 1902 through 1908 for those portions of the design that were outside the focused scope.

Rather, the primary contributors to the SNMPv3 Working Group, and the Working Group in general, devoted their considerable efforts to addressing the missing link -- security and administration -- and in the process made invaluable contributions to the state-of-the-art of management.

They produced a design based on a modular architecture with evolutionary capabilities with emphasis on layering. As a result, SNMPv3 can be thought of as SNMPv2 with additional security and administration capabilities.

In doing so, the Working Group achieved the goal of producing a single specification which has not only the endorsement of the IETF but also has security and administration.

6 SNMPv3 Framework Module Specifications

The specification of the SNMPv3 Management Framework is partitioned in a modular fashion among several documents. It is the intention of the SNMPv3 Working Group that, with proper care, any or all of the individual documents can be revised, upgraded, or replaced as requirements change, new understandings are obtained, and new technologies become available.

Whenever feasible, the initial document set which defines the SNMPv3 Management Framework leverages prior investments defining and implementing the SNMPv2 Management Framework by incorporating by reference each of the specifications of the SNMPv2 Management Framework.

The SNMPv3 Framework augments those specifications with specifications for security and administration for SNMPv3.

The documents which specify the SNMPv3 Management Framework follow the same architecture as those of the prior versions and can be organized for expository purposes into four main categories as follows:

* the data definition language,

* Management Information Base (MIB) modules,

* protocol operations, and

* security and administration.

The first three sets of documents are incorporated from SNMPv2. The fourth set of documents are new to SNMPv3, but, as described previously, build on significant prior related works.

6.1 Data Definition Language

The specifications of the data definition language includes STD 58, RFC 2578, "Structure of Management Information Version 2 (SMIv2)" [26], and related specifications. These documents are updates of RFCs 1902 - 1904 [4-6] which have evolved independently from the other parts of the framework and were republished as STD 58, RFCs 2578 - 2580 [26-28] when promoted from Draft Standard.

The Structure of Management Information (SMIv2) defines fundamental data types, an object model, and the rules for writing and revising MIB modules. Related specifications include STD 58, RFCs 2579, 2580. The updated data definition language is sometimes referred to as SMIv2.

STD 58, RFC 2579, "Textual Conventions for SMIv2" [27], defines an initial set of shorthand abbreviations which are available for use within all MIB modules for the convenience of human readers and writers.

STD 58, RFC 2580, "Conformance Statements for SMIv2" [28], defines the format for compliance statements which are used for describing requirements for agent implementations and capability statements which can be used to document the characteristics of particular implementations.

6.2 MIB Modules

MIB modules usually contain object definitions, may contain definitions of event notifications, and sometimes include compliance statements specified in terms of appropriate object and event notification groups. As such, MIB modules define the management information maintained by the instrumentation in managed nodes, made remotely accessible by management agents, conveyed by the management protocol, and manipulated by management applications.

MIB modules are defined according the rules defined in the documents which specify the data definition language, principally the SMI as supplemented by the related specifications.

There is a large and growing number of standards-based MIB modules, as defined in the periodically updated list of standard protocols [STD 1, RFC 2400]. As of this writing, there are nearly 100 standards-based MIB modules with a total number of defined objects approaching 10,000. In addition, there is an even larger and growing

number of enterprise-specific MIB modules defined unilaterally by various vendors, research groups, consortia, and the like resulting in an unknown and virtually uncountable number of defined objects.

In general, management information defined in any MIB module, regardless of the version of the data definition language used, can be used with any version of the protocol. For example, MIB modules defined in terms of the SNMPv1 SMI (SMIv1) are compatible with the SNMPv3 Management Framework and can be conveyed by the protocols specified therein. Furthermore, MIB modules defined in terms of the SNMPv2 SMI (SMIv2) are compatible with SNMPv1 protocol operations and can be conveyed by it. However, there is one noteworthy exception:

the Counter64 datatype which can be defined in a MIB module defined in SMIv2 format but which cannot be conveyed by an SNMPv1 protocol engine.

6.3 Protocol Operations and Transport Mappings

The specifications for the protocol operations and transport mappings of the SNMPv3 Framework are incorporated by reference to the two SNMPv2 Framework documents.

The specification for protocol operations is found in RFC 1905, "Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2)" [7]. The SNMPv3 Framework is designed to allow various portions of the architecture to evolve independently. For example, it might be possible for a new specification of protocol operations to be defined within the Framework to allow for additional protocol operations.

The specification of transport mappings is found in RFC 1906, "Transport Mappings for Version 2 of the Simple Network Management Protocol (SNMPv2)" [8].

6.4 SNMPv3 Security and Administration

The SNMPv3 document series defined by the SNMPv3 Working Group consists of seven documents at this time:

RFC 2570, "Introduction to Version 3 of the Internet-standard Network Management Framework", which is this document.

RFC 2571, "An Architecture for Describing SNMP Management Frameworks" [15], describes the overall architecture with special emphasis on the architecture for security and administration.

RFC 2572, "Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)" [16], describes the possibly multiple message processing models and the dispatcher portion that can be a part of an SNMP protocol engine.

RFC 2573, "SNMP Applications" [17], describes the five types of

applications that can be associated with an SNMPv3 engine and their elements of procedure.

RFC 2574, "The User-Based Security Model for Version 3 of the Simple Network Management Protocol (SNMPv3)" [18], describes the threats, mechanisms, protocols, and supporting data used to provide SNMP message-level security.

RFC 2575, "View-based Access Control Model for the Simple Network Management Protocol (SNMP)" [19], describes how view-based access control can be applied within command responder and notification originator applications.

The Work in Progress, "Coexistence between Version 1, Version 2, and Version 3 of the Internet-standard Network Management Framework" [20], describes coexistence between the SNMPv3 Management Framework, the SNMPv2 Management Framework, and the original SNMPv1 Management Framework.

7 Document Summaries

The following sections provide brief summaries of each document with slightly more detail than is provided in the overviews above.

7.1 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. These modules are written in the SNMP MIB module language, which contains elements of OSI's Abstract Syntax Notation One (ASN.1) [11] language. STD 58, RFCs 2578, 2579, 2580, together define the MIB module language, specify the base data types for objects, specify a core set of short-hand specifications for data types called textual conventions, and specify a few administrative assignments of object identifier (OID) values.

The SMI is divided into three parts: module definitions, object definitions, and notification definitions.

(1) Module definitions are used when describing information modules. An ASN.1 macro, MODULE-IDENTITY, is used to convey concisely the semantics of an information module.

(2) Object definitions are used when describing managed objects. An ASN.1 macro, OBJECT-TYPE, is used to convey concisely the syntax and semantics of a managed object.

(3) Notification definitions are used when describing unsolicited transmissions of management information. An ASN.1 macro, NOTIFICATION-TYPE, is used to convey concisely the syntax and semantics of a notification.

7.1.1 Base SMI Specification

STD 58, RFC 2578 specifies the base data types for the MIB module language, which include: Integer32, enumerated integers, Unsigned32, Gauge32, Counter32, Counter64, TimeTicks, INTEGER, OCTET STRING, OBJECT IDENTIFIER, IpAddress, Opaque, and BITS. It also assigns values to several object identifiers. STD 58, RFC 2578 further defines the following constructs of the MIB module language:

* IMPORTS to allow the specification of items that are used in a MIB module, but defined in another MIB module.

* MODULE-IDENTITY to specify for a MIB module a description and administrative information such as contact and revision history.

* OBJECT-IDENTITY and OID value assignments to specify a an OID value.

* OBJECT-TYPE to specify the data type, status, and the semantics of managed objects.

* SEQUENCE type assignment to list the columnar objects in a table.

* NOTIFICATION-TYPE construct to specify an event notification.

7.1.2 Textual Conventions

When designing a MIB module, it is often useful to specify in a short-hand way the semantics for a set of objects with similar behavior. This is done by defining a new data type using a base data type specified in the SMI. Each new type has a different name, and specifies a base type with more restrictive semantics. These newly defined types are termed textual conventions, and are used for the convenience of humans reading a MIB module and potentially by "intelligent" management applications. It is the purpose of STD 58, RFC 2579, Textual Conventions for SMIv2 [27], to define the construct, TEXTUAL-CONVENTION, of the MIB module language used to define such new types and to specify an initial set of textual conventions available to all MIB modules.

7.1.3 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 STD 58, RFC 2580, Conformance Statements for SMIv2 [28], to define the constructs of the MIB module language used for these purposes. There are two kinds of constructs:

(1) Compliance statements are used when describing requirements for agents with respect to object and event notification definitions. The MODULE-COMPLIANCE construct is used to convey concisely such requirements.

(2) Capability statements are used when describing capabilities of agents with respect to object and event notification definitions. The AGENT-CAPABILITIES construct is used to convey concisely such capabilities.

Finally, collections of related objects and collections of related event notifications are grouped together to form a unit of conformance. The OBJECT-GROUP construct is used to convey concisely the objects in and the semantics of an object group. The NOTIFICATION-GROUP construct is used to convey concisely the event notifications in and the semantics of an event notification group.

7.2 Protocol Operations

The management protocol provides for the exchange of messages which convey management information between the agents and the management stations. The form of these messages is a message "wrapper" which encapsulates a Protocol Data Unit (PDU).

It is the purpose of RFC 1905, Protocol Operations for SNMPv2 [7], to define the operations of the protocol with respect to the sending and receiving of the PDUs.

7.3 Transport Mappings

SNMP Messages may be used over a variety of protocol suites. It is the purpose of RFC 1906, Transport Mappings for SNMPv2 [8], to define how SNMP messages maps onto an initial set of transport domains. Other mappings may be defined in the future.

Although several mappings are defined, the mapping onto UDP is the preferred mapping. As such, to provide for the greatest level of interoperability, systems which choose to deploy other mappings should also provide for proxy service to the UDP mapping.

7.4 Protocol Instrumentation

It is the purpose of RFC 1907, the Management Information Base for SNMPv2 document [9] to define managed objects which describe the behavior of an SNMPv2 entity.

7.5 Architecture / Security and Administration

It is the purpose of RFC 2571, "An Architecture for Describing SNMP Management Frameworks" [15], to define an architecture for specifying SNMP Management Frameworks. While addressing general architectural issues, it focuses on aspects related to security and administration. It defines a number of terms used throughout the SNMPv3 Management Framework and, in so doing, clarifies and extends the naming of

* engines and applications,
* entities (service providers such as the engines in agents and managers),
* identities (service users), and
* management information, including support for multiple logical contexts.

The document contains a small MIB module which is implemented by all authoritative SNMPv3 protocol engines.

7.6 Message Processing and Dispatch (MPD)

RFC 2572, "Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)" [16], describes the Message Processing and Dispatching for 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.

It is expected that an SNMPv3 protocol engine MUST support at least one Message Processing Model. An SNMPv3 protocol engine MAY support more than one, for example in a multi-lingual system which provides simultaneous support of SNMPv3 and SNMPv1 and/or SNMPv2c.

7.7 SNMP Applications

It is the purpose of RFC 2573, "SNMP Applications" to describe the five types of applications which can be associated with an SNMP engine. They are: Command Generators, Command Responders, Notification Originators, Notification Receivers, and Proxy Forwarders.

The document also defines MIB modules for specifying targets of management operations (including notifications), for notification filtering, and for proxy forwarding.

7.8 User-based Security Model (USM)

RFC 2574, the "User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)" describes the User-based Security Model for SNMPv3. It defines the Elements of Procedure for providing SNMP message-level security.

The document describes the two primary and two secondary threats which are defended against by the User-based Security Model. They are: modification of information, masquerade, message stream modification, and disclosure.

The USM utilizes MD5 [21] and the Secure Hash Algorithm [22] as keyed hashing algorithms [23] for digest computation to provide data integrity

* to directly protect against data modification attacks,

* to indirectly provide data origin authentication, and

* to defend against masquerade attacks.

The USM uses loosely synchronized monotonically increasing time indicators to defend against certain message stream modification attacks. Automatic clock synchronization mechanisms based on the protocol are specified without dependence on third-party time sources and concomitant security considerations.

The USM uses the Data Encryption Standard (DES) [24] in the cipher block chaining mode (CBC) if disclosure protection is desired. Support for DES in the USM is optional, primarily because export and usage restrictions in many countries make it difficult to export and use products which include cryptographic technology.

The document also includes a MIB suitable for remotely monitoring and managing the configuration parameters for the USM, including key distribution and key management.

An entity may provide simultaneous support for multiple security models as well as multiple authentication and privacy protocols. All of the protocols used by the USM are based on pre-placed keys, i.e., private key mechanisms. The SNMPv3 architecture permits the use of asymmetric mechanisms and protocols (commonly called "public key cryptography") but as of this writing, no such SNMPv3 security models utilizing public key cryptography have been published.

7.9 View-based Access Control (VACM)

The purpose of RFC 2575, the "View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)" is to describe the View-based Access Control Model for use in the SNMP architecture. The VACM can simultaneously be associated in a single engine implementation with multiple Message Processing Models and multiple Security Models.

It is architecturally possible to have multiple, different, Access Control Models active and present simultaneously in a single engine implementation, but this is expected to be *_very_* rare in practice and *_far_* less common than simultaneous support for multiple Message Processing Models and/or multiple Security Models.

7.10 SNMPv3 Coexistence and Transition

The purpose of "Coexistence between Version 1, Version 2, and Version 3 of the Internet-standard Network Management Framework" is to describe coexistence between the SNMPv3 Management Framework, the SNMPv2 Management Framework, and the original SNMPv1 Management Framework. In particular, this document describes four aspects of coexistence:

* Conversion of MIB documents from SMIv1 to SMIv2 format

* Mapping of notification parameters

* Approaches to coexistence between entities which support the various versions of SNMP in a multi-lingual network, in particular the processing of protocol operations in multi-lingual implementations, as well as behavior of proxy implementations

* The SNMPv1 Message Processing Model and Community-Based Security Model, which provides mechanisms for adapting SNMPv1 and SNMPv2c into the View-Based Access Control Model (VACM) [19]

8 Security Considerations

As this document is primarily a roadmap document, it introduces no

new security considerations. The reader is referred to the relevant sections of each of the referenced documents for information about security considerations.

9 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

Russ Mundy
TIS Labs at Network Associates
3060 Washington Rd
Glenwood, MD 21738
USA
Phone: +1 301 854 6889
EMail: mundy@tislabs.com

David Partain
Ericsson Radio Systems
Research and Innovation
P.O. Box 1248
SE-581 12 Linkoping
Sweden
Phone: +46 13 28 41 44
EMail: David.Partain@ericsson.com

Bob Stewart
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
U.S.A.
Phone: +1 603 654 6923
EMail: bstewart@cisco.com

10 References

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

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

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

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

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

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

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

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

[9] SNMPv2 Working Group, Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Management Information Base for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1907, January 1996.

[10] SNMPv2 Working Group, Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Coexistence between Version 1 and Version 2 of the Internet-standard Network Management Framework", RFC 1908, January 1996.

[11] Information processing systems - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1), International Organization for Standardization. International Standard 8824, (December, 1987).

[12] McCloghrie, K. and M. Rose, "Management Information Base for Network Management of TCP/IP-based Internets", RFC 1066, August 1988.

[13] McCloghrie, K. and M. Rose, "Management Information Base for Network Management of TCP/IP-based internets: MIB-II, STD 17, RFC 1213, March 1991.

[14] Cerf, V., "IAB Recommendations for the Development of Internet Network Management Standards", RFC 1052, April 1988.

[15] Harrington, D., Presuhn, R. and B. Wijnen, "An Architecture for Describing SNMP Management Frameworks", RFC 2571, April 1999.

[16] Case, J., Harrington, D., Presuhn, R. and B. Wijnen, "Message Processing and Dispatching for the Simple Network Management

Protocol (SNMP)", RFC 2572, April 1999.

[17] Levi, D., Meyer, P. and B. Stewart, "SNMP Applications", RFC 2573, April 1999.

[18] Blumenthal, U. and B. Wijnen, "The User-Based Security Model for Version 3 of the Simple Network Management Protocol (SNMPv3)", RFC 2574, April 1999.

[19] Wijnen, B., Presuhn, R. and K. McCloghrie, "View-based Access Control Model for the Simple Network Management Protocol (SNMP)", RFC 2575, April 1999.

[20] 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", Work in Progress.

[21] Rivest, R., "Message Digest Algorithm MD5", RFC 1321, April 1992.

[22] 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)

[23] Krawczyk, H., Bellare, M. and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[24] 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).

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

[26] 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.

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

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

11 Full Copyright Statement

Copyright (C) The Internet Society (1998). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING

BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE."

Acknowledgement

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

Network Working Group
Request for Comments: 2571
Obsoletes: 2271
Category: Standards Track

D. Harrington
Cabletron Systems, Inc.
R. Presuhn
BMC Software, Inc.
B. Wijnen
IBM T. J. Watson Research
April 1999

An Architecture for Describing SNMP Management Frameworks

Status of this Memo

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

Copyright Notice

Copyright (C) The Internet Society (1999). All Rights Reserved.

Abstract

This document describes an architecture for describing 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.

Table of Contents

1. Introduction 4
1.1. Overview 4
1.2. SNMP 4
1.3. Goals of this Architecture 5
1.4. Security Requirements of this Architecture 6
1.5. Design Decisions 7
2. Documentation Overview 9
2.1. Document Roadmap 10
2.2. Applicability Statement 10
2.3. Coexistence and Transition 10
2.4. Transport Mappings 11
2.5. Message Processing 11

2.6. Security 11
2.7. Access Control 12
2.8. Protocol Operations 12
2.9. Applications 13
2.10. Structure of Management Information 14
2.11. Textual Conventions 14
2.12. Conformance Statements 14
2.13. Management Information Base Modules 14
2.13.1. SNMP Instrumentation MIBs 14
2.14. SNMP Framework Documents 14
3. Elements of the Architecture 15
3.1. The Naming of Entities 16
3.1.1. SNMP engine 17
3.1.1.1. snmpEngineID 17
3.1.1.2. Dispatcher 17
3.1.1.3. Message Processing Subsystem 18
3.1.1.3.1. Message Processing Model 18
3.1.1.4. Security Subsystem 18
3.1.1.4.1. Security Model 19
3.1.1.4.2. Security Protocol 19
3.1.2. Access Control Subsystem 19
3.1.2.1. Access Control Model 20
3.1.3. Applications 20
3.1.3.1. SNMP Manager 20
3.1.3.2. SNMP Agent 22
3.2. The Naming of Identities 23
3.2.1. Principal 23
3.2.2. securityName 23
3.2.3. Model-dependent security ID 24
3.3. The Naming of Management Information 25
3.3.1. An SNMP Context 26
3.3.2. contextEngineID 26
3.3.3. contextName 27
3.3.4. scopedPDU 27
3.4. Other Constructs 27
3.4.1. maxSizeResponseScopedPDU 27
3.4.2. Local Configuration Datastore 27
3.4.3. securityLevel 27
4. Abstract Service Interfaces 28
4.1. Dispatcher Primitives 28
4.1.1. Generate Outgoing Request or Notification 28
4.1.2. Process Incoming Request or Notification PDU 29
4.1.3. Generate Outgoing Response 29
4.1.4. Process Incoming Response PDU 29
4.1.5. Registering Responsibility for Handling SNMP PDUs 30
4.2. Message Processing Subsystem Primitives 30
4.2.1. Prepare Outgoing SNMP Request or Notification Message ... 31
4.2.2. Prepare an Outgoing SNMP Response Message 31

4.2.3. Prepare Data Elements from an Incoming SNMP Message 32
4.3. Access Control Subsystem Primitives 32
4.4. Security Subsystem Primitives 33
4.4.1. Generate a Request or Notification Message 33

4.4.2. Process Incoming Message 33
4.4.3. Generate a Response Message 34
4.5. Common Primitives 34
4.5.1. Release State Reference Information 35
4.6. Scenario Diagrams 36
4.6.1. Command Generator or Notification Originator 36
4.6.2. Scenario Diagram for a Command Responder Application 37
5. Managed Object Definitions for SNMP Management Frameworks ... 38
6. IANA Considerations 48
6.1. Security Models 48
6.2. Message Processing Models 48
6.3. SnmpEngineID Formats 49
7. Intellectual Property 49
8. Acknowledgements 49
9. Security Considerations 51
10. References 52
11. Editor's Addresses 54
A. Guidelines for Model Designers 55
A.1. Security Model Design Requirements 55
A.1.1. Threats 55
A.1.2. Security Processing 56
A.1.3. Validate the security-stamp in a received message 56
A.1.4. Security MIBs 57
A.1.5. Cached Security Data 57
A.2. Message Processing Model Design Requirements 57
A.2.1. Receiving an SNMP Message from the Network 58
A.2.2. Sending an SNMP Message to the Network 58
A.3. Application Design Requirements 59
A.3.1. Applications that Initiate Messages 59
A.3.2. Applications that Receive Responses 59
A.3.3. Applications that Receive Asynchronous Messages 60
A.3.4. Applications that Send Responses 60
A.4. Access Control Model Design Requirements 60
B. Full Copyright Statement 62

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.

```
+------------------------- Document Set ----------------------------+
|                                                                    |
| +----------+              +-----------------+  +----------------+ |
| | Document |              | Applicability * |  | Coexistence    | |
| | Roadmap  |              | Statement       |  | & Transition   | |
| +----------+              +-----------------+  +----------------+ |
|                                                                    |
| +------------------------------------------------------------------+ |
| | Message Handling                                                 | |
| | +----------------+  +-----------------+  +-----------------+  | |
| | | Transport      |  | Message         |  | Security        |  | |
| | | Mappings       |  | Processing and  |  |                 |  | |
| | |                |  | Dispatcher      |  |                 |  | |
| | +----------------+  +-----------------+  +-----------------+  | |
| +------------------------------------------------------------------+ |
|                                                                    |
| +------------------------------------------------------------------+ |
| | PDU Handling                                                     | |
| | +----------------+  +-----------------+  +-----------------+  | |
| | | Protocol       |  | Applications    |  | Access          |  | |
| | | Operations     |  |                 |  | Control         |  | |
| | +----------------+  +-----------------+  +-----------------+  | |
| +------------------------------------------------------------------+ |
|                                                                    |
```

```
| +-----------------------------------------------------------------+ |
| | Information Model                                               | |
| | +--------------+   +--------------+    +---------------+       | |
| | | Structure of |   | Textual      |    | Conformance   |       | |
| | | Management   |   | Conventions  |    | Statements    |       | |
| | | Information  |   |              |    |               |       | |
| | +--------------+   +--------------+    +---------------+       | |
| +-----------------------------------------------------------------+ |
|                                                                     |
| +-----------------------------------------------------------------+ |
| | MIB Modules written in various formats, e.g.:                   | |
| | +-------------+ +-------------+ +----------+ +----------+       | |
| | | Standard v1 | | Standard v1 | | Historic | | Draft v2 |       | |
| | | RFC 1157    | | RFC 1212    | | RFC 14xx | | RFC 19xx |       | |
| | | format      | | format      | | format   | | format   |       | |
| | +-------------+ +-------------+ +----------+ +----------+       | |
| +-----------------------------------------------------------------+ |
|                                                                     |
+---------------------------------------------------------------------+
```


Those marked with an asterisk (*) are expected to be written in the future. 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 to Version 3 of the Internet-standard Network Management Framework" [RFC2570].

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 coexistance document is [SNMP-COEX], "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, RFC 1905 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, RFC 1905 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, RFC 1905 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, RFC 1905 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, RFC 1905 defines the following operations 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, RFC 1905 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, RFC 1905 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.

Applications documents describe 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 remote configuration of the Model.

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 RFCs 1905-1907. 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 RFC 1901. 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.

```
+-------------------------------------------------------------------+
|  SNMP entity                                                      |
|                                                                   |
|  +-------------------------------------------------------------+  |
|  |  SNMP engine (identified by snmpEngineID)                   |  |
|  |                                                             |  |
|  |  +------------+ +------------+ +-----------+ +-----------+  |  |
|  |  |            | |            | |           | |           |  |  |
|  |  | Dispatcher | | Message    | | Security  | | Access    |  |  |
|  |  |            | | Processing | | Subsystem | | Control   |  |  |
|  |  |            | | Subsystem  | |           | | Subsystem |  |  |
|  |  |            | |            | |           | |           |  |  |
|  |  +------------+ +------------+ +-----------+ +-----------+  |  |
|  |                                                             |  |
|  +-------------------------------------------------------------+  |
|                                                                   |
|  +-------------------------------------------------------------+  |
|  |  Application(s)                                             |  |
|  |                                                             |  |
|  |  +-------------+  +--------------+  +--------------+        |  |
|  |  | Command     |  | Notification |  | Proxy        |        |  |
|  |  | Generator   |  | Receiver     |  | Forwarder    |        |  |
|  |  +-------------+  +--------------+  +--------------+        |  |
|  |                                                             |  |
|  |  +-------------+  +--------------+  +--------------+        |  |
|  |  | Command     |  | Notification |  | Other        |        |  |
|  |  | Responder   |  | Originator   |  |              |        |  |
|  |  +-------------+  +--------------+  +--------------+        |  |
|  |                                                             |  |
|  +-------------------------------------------------------------+  |
|                                                                   |
+-------------------------------------------------------------------+
```

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.

```
+-------------------------------------------------------------------+
|                                                                   |
|  Message Processing Subsystem                                     |
|                                                                   |
|  +------------+  +------------+  +------------+  +------------+  |
|  |          * |  |          * |  |          * |  |          * |  |
|  | SNMPv3     |  | SNMPv1     |  | SNMPv2c    |  | Other      |  |
|  | Message    |  | Message    |  | Message    |  | Message    |  |
|  | Processing |  | Processing |  | Processing |  | Processing |  |
|  | Model      |  | Model      |  | Model      |  | Model      |  |
|  |            |  |            |  |            |  |            |  |
|  +------------+  +------------+  +------------+  +------------+  |
|                                                                   |
+-------------------------------------------------------------------+
```

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.

```
+-------------------------------------------------------------------+
|                                                                   |
|  Security Subsystem                                               |
|                                                                   |
|  +----------------+  +-----------------+  +--------------------+ |
|  |              * |  |               * |  |                  * | |
```

```
|  | User-Based      |  | Other           |  | Other              |  |
|  | Security        |  | Security        |  | Security           |  |
|  | Model           |  | Model           |  | Model              |  |
|  |                 |  |                 |  |                    |  |
|  +----------------+  +-----------------+  +-------------------+  |
|                                                                    |
+--------------------------------------------------------------------+
```

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.

```
+--------------------------------------------------------------------+
|                                                                    |
|  Access Control Subsystem                                          |
|                                                                    |
|  +---------------+   +-----------------+   +-------------------+  |
|  |             * |   |               * |   |                 * |  |
|  | View-Based    |   | Other           |   | Other             |  |
|  | Access        |   | Access          |   | Access            |  |
|  | Control       |   | Control         |   | Control           |  |
|  | Model         |   | Model           |   | Model             |  |
|  |               |   |                 |   |                   |  |
|  +---------------+   +-----------------+   +-------------------+  |
|                                                                    |
+--------------------------------------------------------------------+
```


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.

* One or more models may be present.

```
                    (traditional SNMP manager)
+-------------------------------------------------------------------+
| +--------------+  +--------------+  +--------------+  SNMP entity |
| | NOTIFICATION |  | NOTIFICATION |  |   COMMAND    |              |
| |  ORIGINATOR  |  |   RECEIVER   |  |  GENERATOR   |              |
| | applications |  | applications |  | applications |              |
| +--------------+  +--------------+  +--------------+              |
|         ^                ^                 ^                      |
|         |                |                 |                      |
|         v                v                 v                      |
|         +-------+--------+-----------------+                      |
|                 ^                                                 |
|                 |     +---------------------+  +----------------+ |
|                 |     | Message Processing  |  | Security       | |
```

```
| Dispatcher      v     | Subsystem           |   | Subsystem      | |
| +-------------------+ |     +-------------+  |   |                | |
| | PDU Dispatcher    | |  +->| v1MP     * |<--->| +------------+ | |
| |                   | |  |  +-------------+  |   | | Other      | | |
| |                   | |  |  +-------------+  |   | | Security   | | |
| |                   | |  +->| v2cMP    * |<--->| | Model      | | |
| | Message           | |  |  +-------------+  |   | +------------+ | |
| | Dispatcher  <---------->+                   |   |                | |
| |                   | |  |  +-------------+  |   | +------------+ | |
| |                   | |  +->| v3MP     * |<--->| | User-based | | |
| | Transport         | |  |  +-------------+  |   | | Security   | | |
| | Mapping           | |  |  +-------------+  |   | | Model      | | |
| | (e.g RFC1906)     | |  +->| otherMP  * |<--->| +------------+ | |
| +-------------------+ |     +-------------+  |   |                | |
|          ^            +----------------------+  +----------------+ |
|          |                                                          |
|          v                                                          |
+----------------------------------------------------------------------+
+-----+ +-----+       +-------+
| UDP | | IPX | . . . | other |
+-----+ +-----+       +-------+
   ^       ^              ^
   |       |              |
   v       v              v
+-------------------------------+
|           Network             |
+-------------------------------+
```


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.

```
+-------------------------------+
|           Network             |
+-------------------------------+
   ^       ^              ^
   |       |              |
   v       v              v
+-----+ +-----+       +-------+
| UDP | | IPX | . . . | other |
+-----+ +-----+       +-------+              (traditional SNMP agent)
+----------------------------------------------------------------------+
|              ^                                                      |
|              |        +----------------------+  +----------------+ |
|              |        | Message Processing   |  | Security       | |
| Dispatcher   v        | Subsystem            |  | Subsystem      | |
| +-------------------+ |     +-------------+  |  |                | |
```

```
| | Transport          | |  +->| v1MP     * |<--->| +------------+ | |
| | Mapping            | |  |  +------------+  |  | | Other      | | |
| | (e.g. RFC1906)     | |  |  +------------+  |  | | Security   | | |
| |                    | |  +->| v2cMP    * |<--->| | Model      | | |
| | Message            | |  |  +------------+  |  | +------------+ | |
| | Dispatcher  <--------->|  +------------+  |  | +------------+ | |
| |                    | |  +->| v3MP     * |<--->| | User-based | | |
| |                    | |  |  +------------+  |  | | Security   | | |
| | PDU Dispatcher     | |  |  +------------+  |  | | Model      | | |
| +--------------------+ |  +->| otherMP  * |<--->| +------------+ | |
|              ^         |     +------------+  |  |                | |
|              |         +---------------------+  +----------------+ |
|              v                                                      |
|      +-------+-------------------------+---------------+           |
|      ^                                 ^               ^           |
|      |                                 |               |           |
|      v                                 v               v           |
| +-------------+   +---------+   +--------------+  +-------------+ |
| |   COMMAND   |   | ACCESS  |   | NOTIFICATION |  |    PROXY  * | |
| |  RESPONDER  |<->| CONTROL |<->|  ORIGINATOR  |  |  FORWARDER  | |
| | application |   |         |   | applications |  | application | |
| +-------------+   +---------+   +--------------+  +-------------+ |
|      ^                                 ^                            |
|      |                                 |                            |
|      v                                 v                            |
| +-------------------------------------------+                       |
| |             MIB instrumentation           |      SNMP entity |
+-----------------------------------------------------------------+
```


3.2. The Naming of Identities

```
                          principal
                              ^
                              |
                              |
+----------------------------|-------------+
| SNMP engine                v             |
|                     +--------------+     |
|                     |              |     |
|  +-----------------| securityName |---+ |
|  | Security Model  |              |   | |
|  |                 +--------------+   | |
|  |                         ^          | |
|  |                         |          | |
|  |                         v          | |
|  |  +-----------------------------+  | |
|  |  |                             |  | |
|  |  | Model                       |  | |
|  |  | Dependent                   |  | |
|  |  | Security ID                 |  | |
|  |  |                             |  | |
|  |  +-----------------------------+  | |
|  |                         ^          | |
```

```
|  |                          |          |  |
|  +--------------------------|----------+  |
|                             |             |
|                             |             |
+-----------------------------|-------------+
                              |
                              v
                           network
```

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, example: abcd)        |
|                                                                 |
|  +-------------------------------------------------------------+ |
|  | SNMP engine (identified by snmpEngineID)                    | |
|  |                                                             | |
|  | +-------------+ +------------+ +-----------+ +-----------+ | |
|  | |             | |            | |           | |           | | |
|  | | Dispatcher  | | Message    | | Security  | | Access    | | |
|  | |             | | Processing | | Subsystem | | Control   | | |
|  | |             | | Subsystem  | |           | | Subsystem | | |
|  | |             | |            | |           | |           | | |
|  | +-------------+ +------------+ +-----------+ +-----------+ | |
|  |                                                             | |
|  +-------------------------------------------------------------+ |
|                                                                 |
|  +-------------------------------------------------------------+ |
|  |  Command Responder Application                              | |
|  |  (contextEngineID, example: abcd)                           | |
|  |                                                             | |
|  |  example contextNames:                                      | |
|  |                                                             | |
|  |  "bridge1"          "bridge2"            "" (default)       | |
|  |  ---------          ---------            ------------       | |
|  |      |                  |                    |              | |
```

```
|  +------|------------------|-------------------|--------------+ |
|         |                  |                   |               |
|  +------|------------------|-------------------|--------------+ |
|  |  MIB | instrumentation  |                   |              | |
|  |  +---v------------+ +---v------------+ +----v-----------+  | |
|  |  | context        | | context        | | context        |  | |
|  |  |                | |                | |                |  | |
|  |  | +------------+ | | +------------+ | | +------------+ |  | |
|  |  | | bridge MIB | | | | bridge MIB | | | | 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 [RFC2233], 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, RFC1905 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            -- identification of the principal
  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.

```
Command            Dispatcher                Message            Security
Generator             |                      Processing            Model
|                     |                      Model                     |
|      sendPdu        |                         |                      |
|------------------->|                         |                      |
|                     | prepareOutgoingMessage |                      |
:                     |------------------------>|                      |
:                     |                         | generateRequestMsg   |
```

```
:                       |                          |-------------------->|
:                       |                          |                     |
:                       |                          |<--------------------|
:                       |                          |                     |
:                       |<-------------------------|                     |
:                       |                          |                     |
:                       |------------------+       |                     |
:                       | Send SNMP        |       |                     |
:                       | Request Message  |       |                     |
:                       | to Network       |       |                     |
:                       |                  v       |                     |
:                       :                  :       :                     :
:                       :                  :       :                     :
:                       :                  :       :                     :
:                       |                  |       |                     |
:                       | Receive SNMP     |       |                     |
:                       | Response Message |       |                     |
:                       | from Network     |       |                     |
:                       |<-----------------+       |                     |
:                       |                          |                     |
:                       |   prepareDataElements    |                     |
:                       |------------------------->|                     |
:                       |                          | processIncomingMsg  |
:                       |                          |-------------------->|
:                       |                          |                     |
:                       |                          |<--------------------|
:                       |                          |                     |
:                       |<-------------------------|                     |
:                       |                          |                     |
| processResponsePdu    |                          |                     |
|<--------------------|                          |                     |
|                       |                          |                     |
```


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 a SNMP message is received, and how the Response is (asynchronously) send back to the network.

```
Command                 Dispatcher              Message          Security
Responder                   |                   Processing          Model
|                           |                   Model                  |
|                           |                      |                   |
| registerContextEngineID   |                      |                   |
|------------------------->|                      |                   |
|<-------------------------|                |     |                   |
|                           | Receive SNMP   |     |                   |
:                           | Message        |     |                   |
:                           | from Network   |     |                   |
:                           |<--------------+     |                   |
:                           |                     |                   |
:                           |prepareDataElements  |                   |
```

```
:                           |------------------->|                    |
:                           |                    | processIncomingMsg |
:                           |                    |------------------->|
:                           |                    |                    |
:                           |                    |<-------------------|
:                           |                    |                    |
:                           |<-------------------|                    |
|     processPdu            |                    |                    |
|<--------------------------|                    |                    |
|                           |                    |                    |
:                           :                    :                    :
:                           :                    :                    :
|    returnResponsePdu      |                    |                    |
|-------------------------->|                    |                    |
:                           | prepareResponseMsg |                    |
:                           |------------------->|                    |
:                           |                    |generateResponseMsg |
:                           |                    |------------------->|
:                           |                    |                    |
:                           |                    |<-------------------|
:                           |                    |                    |
:                           |<-------------------|                    |
:                           |                    |                    |
:                           |--------------+     |                    |
:                           | Send SNMP    |     |                    |
:                           | Message      |     |                    |
:                           | to Network   |     |                    |
:                           |              v     |                    |
```


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 "9901190000Z"              -- 19 January 1999
    ORGANIZATION "SNMPv3 Working Group"
    CONTACT-INFO "WG-EMail:   snmpv3@tis.com
                  Subscribe:  majordomo@tis.com
                              In message body:  subscribe snmpv3

                  Chair:      Russ Mundy
                              TIS Labs at Network Associates
                  postal:     3060 Washington Rd
                              Glenwood MD 21738
                              USA
                  EMail:      mundy@tis.com
```

phone: +1 301-854-6889

Co-editor Dave Harrington
Cabletron Systems, Inc.
postal: Post Office Box 5005
Mail Stop: Durham
35 Industrial Way
Rochester, NH 03867-5005
USA
EMail: dbh@ctron.com
phone: +1 603-337-7357

Co-editor Randy Presuhn
BMC Software, Inc.
postal: 965 Stewart Drive
Sunnyvale, CA 94086
USA
EMail: randy_presuhn@bmc.com
phone: +1 408-616-3100

Co-editor: Bert Wijnen
IBM T.J. Watson Research
postal: Schagen 33
3461 GL Linschoten

Netherlands
EMail: wijnen@vnet.ibm.com
phone: +31 348-432-794

```
                "
    DESCRIPTION  "The SNMP Management Architecture MIB"
-- Revision History

    REVISION     "9901190000Z"            -- 19 January 1999
    DESCRIPTION  "Updated editors' addresses, fixed typos.
                  Published as RFC2571.
                "
    REVISION     "9711200000Z"            -- 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 strings 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

                    127-255 - 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
                 securityModel of the Security Subsystem within the
                 SNMP Management Architecture.

                 The values for securityModel are allocated as
                 follows:

                 - The zero value is reserved.
                 - 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 260.

This scheme for allocation of securityModel values allows for a maximum of 255 standards-based Security Models, and for a maximum of 255 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 255 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 a 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 260.

                 This scheme for allocating messageProcessingModel
                 values allows for a maximum of 255 standards-
                 based Message Processing Models, and for a
                 maximum of 255 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 "255a"
        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 [RFC1905].

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.
                "
    ::= { 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 255 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 a 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 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.

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

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

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

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

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

[RFC1905] The SNMPv2 Working Group, Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1905, January 1996.

[RFC1906] The SNMPv2 Working Group, Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Transport Mappings for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1906, January 1996.

[RFC1907] The SNMPv2 Working Group, Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Management Information Base for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1907 January 1996.

[RFC1908] The SNMPv2 Working Group, Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Coexistence between Version 1 and Version 2 of the SNMP-standard Network Management Framework", RFC 1908, 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.

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

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

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

[RFC2233] McCloghrie, K. and F. Kastenholz. "The Interfaces Group MIB using SMIv2", RFC 2233, November 1997.

[RFC2572] Case, J., Harrington, D., Presuhn, R. and B. Wijnen, "Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)", RFC 2572, April 1999.

[RFC2574] Blumenthal, U. and B. Wijnen, "The User-Based Security Model for Version 3 of the Simple Network Management Protocol (SNMPv3)", RFC 2574, April 1999.

[RFC2575] Wijnen, B., Presuhn, R. and K. McCloghrie, "View-based Access Control Model for the Simple Network Management Protocol (SNMP)", RFC 2575, April 1999.

[RFC2573] Levi, D. B., Meyer, P. and B. Stewart, "SNMP Applications", RFC 2573, April 1999.

[RFC2570] Case, J., Mundy, R., Partain, D. and B. Stewart, "Introduction to Version 3 of the Internet-standard Network Management Framework", RFC 2570, April 1999.

[SNMP-COEX] Frye, R., Levi, D. and B. Wijnen, "Coexistence between Version 1, Version 2, and Version 3 of the Internet-standard Network Management Framework", Work in Progress.

11. Editor's Addresses

Bert Wijnen
IBM T.J. Watson Research
Schagen 33
3461 GL Linschoten
Netherlands

Phone: +31 348-432-794
EMail: wijnen@vnet.ibm.com

Dave Harrington
Cabletron Systems, Inc
Post Office Box 5005
Mail Stop: Durham
35 Industrial Way
Rochester, NH 03867-5005
USA

Phone: +1 603-337-7357
EMail: dbh@ctron.com

Randy Presuhn
BMC Software, Inc.
965 Stewart Drive
Sunnyvale, CA 94086

USA

Phone: +1 408-616-3100
Fax: +1 408-616-3101
EMail: randy_presuhn@bmc.com

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.

B. Full Copyright Statement

Copyright (C) The Internet Society (1999). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Acknowledgement

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

Network Working Group
Request for Comments: 2572
Obsoletes: 2272
Category: Standards Track

J. Case
SNMP Research Inc.
D. Harrington
Cabletron Systems, Inc.
R. Presuhn
BMC Software, Inc.
B. Wijnen
IBM T. J. Watson Research
April 1999

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

Status of this Memo

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

Copyright Notice

Copyright (C) The Internet Society (1999). All Rights Reserved.

Abstract

This document describes the Message Processing and Dispatching for SNMP messages within the SNMP architecture [RFC2571]. 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.

Table of Contents

1. Introduction 3
2. Overview 3
2.1. The Dispatcher. 5
2.2. Message Processing Subsystem 5
3. Elements of Message Processing and Dispatching 5
3.1. messageProcessingModel 6
3.2. pduVersion 6
3.3. pduType 7
3.4. sendPduHandle 7
4. Dispatcher Elements of Procedure 7
4.1. Sending an SNMP Message to the Network 7

4.1.1. Sending a Request or Notification 7
4.1.2. Sending a Response to the Network 9
4.2. Receiving an SNMP Message from the Network 11
4.2.1. Message Dispatching of received SNMP Messages 11
4.2.2. PDU Dispatching for Incoming Messages 12
4.2.2.1. Incoming Requests and Notifications 12
4.2.2.2. Incoming Responses 14
4.3. Application Registration for Handling PDU types 15
4.4. Application Unregistration for Handling PDU Types 16
5. Definitions 16
5.1. Definitions for SNMP Message Processing and Dispatching ... 16
6. The SNMPv3 Message Format 20
6.1. msgVersion 21
6.2. msgID 21
6.3. msgMaxSize 21
6.4. msgFlags 22
6.5. msgSecurityModel 24
6.6. msgSecurityParameters 24
6.7. scopedPduData 24
6.8. scopedPDU 25
6.8.1. contextEngineID 25
6.8.2. contextName 25
6.8.3. data 25
7. Elements of Procedure for v3MP 25
7.1. Prepare an Outgoing SNMP Message 36
7.2. Prepare Data Elements from an Incoming SNMP Message 31
8. Intellectual Property 37
9. Acknowledgements 37
10. Security Considerations 39
11. References 40
12. Editors' Addresses 41
13. Changes From RFC 2272 42
14. Full Copyright Statement 44

1. Introduction

The Architecture for describing Internet Management Frameworks [RFC2571] 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 [RFC2571].

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 a 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 RFC 2119.

2. Overview

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

```
+-----------------------------------------------------------------+
| SNMP Entity                                                     |
```

```
|                                                                      |
| +--------------------------------------------------------------+ |
| | Applications                                                  | |
| | +-----------+  +--------------+                               | |
| | | Command   |  | Notification |                               | |
| | | Generator |  | Originator   | +-----------+ +--------------+| |
| | +-----------+  +--------------+ | Proxy     | | Other        | |
| | +-----------+  +--------------+ | Forwarder | |Application(s)|| |
| | | Command   |  | Notification | +-----------+ +--------------+| |
| | | Responder |  | Receiver     |                               | |
| | +-----------+  +--------------+                               | |
| +--------------------------------------------------------------+ |
|        ^                ^              ^           ^              |
|        |                |              |           |              |
|        v                v              v           v              |
|        +--------+-------+---------------+-----------+              |
|                 ^                                                  |
|                 |    +---------------------+  +-----------------+ |
|                 |    | Message Processing  |  | Security        | |
| Dispatcher      v    | Subsystem           |  | Subsystem       | |
| +-------------------+ |     +------------+  |  |                 | |
| | PDU Dispatcher    | |  +->| v1MP     * |<--->| +-------------+ | |
| |                   | |  |  +------------+  |  | | Other       | | |
| |                   | |  |  +------------+  |  | | Security    | | |
| |                   | |  +->| v2cMP    * |<--->| | Model       | | |
| | Message           | |  |  +------------+  |  | +-------------+ | |
| | Dispatcher  <-------->+                  |  |                 | |
| |                   | |  |  +------------+  |  | +-------------+ | |
| |                   | |  +->| v3MP     * |<--->| | User-based  | | |
| | Transport         | |  |  +------------+  |  | | Security    | | |
| | Mapping           | |  |  +------------+  |  | | Model       | | |
| | (e.g RFC 1906)    | |  +->| otherMP  * |<--->| +-------------+ | |
| +-------------------+ |     +------------+  |  |                 | |
|          ^            +---------------------+  +-----------------+ |
|          |                                                         |
+----------|-------------------------------------------------------+
           v
  +-------------------+
  |   Network         |
  +-------------------+
```


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 [RFC2571] 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 [RFC1905]. 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, 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 [RFC1905] elements of procedure and syntax instead of those specified for SNMPv1.

3.3. pduType

A value of pduType represents a specific type of protocol operation. The values of 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 trap.

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 via the transport specified by the transportDomain to the address specified by the transportAddress.

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 over the transport specified by the transportDomain to the address specified by the transportAddress.

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 [RFC1907] 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 [RFC1907] counter is incremented, and the message is discarded without further processing. If the version is not supported, then the snmpInBadVersions [RFC1907] 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.

   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 means for requesting simultaneous registration for multiple values of 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 means for requesting unregistration for simultaneous multiple contextEngineID values, e.g., all contextEngineID values, and may also provide 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 "9905041636Z"               -- 4 April 1999
    ORGANIZATION "SNMPv3 Working Group"
    CONTACT-INFO "WG-EMail:   snmpv3@lists.tislabs.com
                  Subscribe:  majordomo@lists.tislabs.com
                              In message body:  subscribe snmpv3

                  Chair:      Russ Mundy
                              TIS Labs at Network Associates
                  postal:     3060 Washington Road
                              Glenwood, MD 21738
                              USA
                  EMail:      mundy@tislabs.com
                  phone:      +1 301-854-6889
```

```
                    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   Dave Harrington
                                Cabletron Systems, Inc.
                    postal:     Post Office Box 5005
                                MailStop: Durham
                                35 Industrial Way
                                Rochester, NH 03867-5005
                                USA
                    EMail:      dbh@ctron.com
                    phone:      +1 603-337-7357

                    Co-editor:  Randy Presuhn
                                BMC Software, Inc.
                    postal:     965 Stewart Drive
                                Sunnyvale, CA 94086
                                USA
                    EMail:      randy_presuhn@bmc.com
                    phone:      +1 408-616-3100

                    Co-editor:  Bert Wijnen
                                IBM T. J. Watson Research
                    postal:     Schagen 33
                                3461 GL Linschoten
                                Netherlands
                    EMail:      wijnen@vnet.ibm.com
                    phone:      +31 348-432-794

                   "
    DESCRIPTION  "The MIB for Message Processing and Dispatching"
    REVISION     "9905041636Z"              -- 4 April 1999
    DESCRIPTION  "Updated addresses, published as RFC 2572."
    REVISION     "9709300000Z"              -- 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 RFC 1905
    }
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 [RFC2571] 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, include request-type PDUs (such

as a Get PDU) and an 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 [RFC2571] 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 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 [RFC1905].

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 determine 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 reportableFlag is zero, then the original message is discarded, and no further processing is done. A result of FAILURE is returned. SNMPv3 Message Processing is complete.

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

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 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 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 a (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 [RFC1906]) of an SNMPv3Message value, then the snmpInASNParseErrs counter [RFC1907] 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 [RFC1907] 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 RFC 1905, 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 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 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 [RFC2571] 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

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

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

[RFC1905] The SNMPv2 Working Group, Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1905, January 1996.

[RFC1906] The SNMPv2 Working Group, Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Transport Mappings for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1906, January 1996.

[RFC1907] The SNMPv2 Working Group, Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Management Information Base for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1907 January 1996.

[RFC1908] The SNMPv2 Working Group, Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Coexistence between Version 1 and Version 2 of the Internet-standard Network Management Framework", RFC 1908, January 1996.

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

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

[RFC2571] Harrington, D., Presuhn, R. and B. Wijnen, "An Architecture for describing SNMP Management Frameworks", RFC 2571, April 1999.

[RFC2574] Blumenthal, U. and B. Wijnen, "The User-Based Security Model for Version 3 of the Simple Network Management Protocol (SNMPv3)", RFC 2574, April 1999.

[RFC2575] Wijnen, B., Presuhn, R. and K. McCloghrie, "View-based Access Control Model for the Simple Network Management Protocol (SNMP)", RFC 2575, April 1999.

[RFC2573] Levi, D., Meyer, P. and B. Stewart, "SNMP Applications", RFC 2573, April 1999.

[RFC2570] Case, J., Mundy, R., Partain, D. and B. Stewart, "Introduction to Version 3 of the Internet-standard Network Management Framework", RFC 2570, April 1999.

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

Dave Harrington
Cabletron Systems, Inc
Post Office Box 5005
Mail Stop: Durham
35 Industrial Way
Rochester, NH 03867-5005
USA

Phone: +1 603-337-7357
EMail: dbh@ctron.com

Randy Presuhn
BMC Software, Inc.
965 Stewart Drive
Sunnyvale, CA 94086
USA

Phone: +1 408-616-3100
EMail: randy_presuhn@bmc.com

Bert Wijnen
IBM T. J. Watson Research
Schagen 33
3461 GL Linschoten
Netherlands

Phone: +31 348-432-794
EMail: wijnen@vnet.ibm.com

13. Changes From RFC 2272

The following change log records major changes from the previous version of this document, RFC 2272.

- Updated contact information for editors.

- Made parameter identification in prepareResponseMessage()

consistent, both internally and with architecture.

- Made references to processIncomingMsg() consistent, both internally and with architecture.

- Deleted superfluous expectResponse parameter from processIncomingMsg(), consistent with architecture.

- Fixed typos.

- Removed sending of a report PDU from step 4 on page 30 on RFC 2272.

- Use "PDU Class" terminology instead of directly using RFC 1905 PDU types, in order to potentially allow use of new PDU types in the future.

- Added intro document to references.

- Made various clarifications to the text.

- The handling of the reportableFlag has been made consistent.

- The acknowledgement list has been updated.

14. Full Copyright Statement

Copyright (C) The Internet Society (1999). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Acknowledgement

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

Network Working Group
Request for Comments: 2573
Obsoletes: 2273
Category: Standards Track

D. Levi
SNMP Research, Inc.
P. Meyer
Secure Computing Corporation
B. Stewart
Cisco Systems
April 1999

SNMP Applications

Status of this Memo

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

Copyright Notice

Copyright (C) The Internet Society (1999). All Rights Reserved.

Abstract

This memo describes five types of SNMP applications which make use of an SNMP engine as described in [RFC2571]. The types of application described are Command Generators, Command Responders, Notification Originators, Notification Receivers, and Proxy Forwarders.

This memo also defines MIB modules for specifying targets of management operations, for notification filtering, and for proxy forwarding.

Table Of Contents

1 Overview 2
1.1 Command Generator Applications 3
1.2 Command Responder Applications 3
1.3 Notification Originator Applications 3
1.4 Notification Receiver Applications 3
1.5 Proxy Forwarder Applications 4
2 Management Targets 5
3 Elements Of Procedure 6
3.1 Command Generator Applications 6
3.2 Command Responder Applications 9
3.3 Notification Originator Applications 14
3.4 Notification Receiver Applications 17

3.5 Proxy Forwarder Applications 19
3.5.1 Request Forwarding 20
3.5.1.1 Processing an Incoming Request 20
3.5.1.2 Processing an Incoming Response 23
3.5.1.3 Processing an Incoming Internal-Class PDU 24
3.5.2 Notification Forwarding 25
4 The Structure of the MIB Modules 28
4.1 The Management Target MIB Module 28
4.1.1 Tag Lists 29
4.1.2 Definitions 30
4.2 The Notification MIB Module 43
4.2.1 Definitions 43
4.3 The Proxy MIB Module 55
4.3.1 Definitions 55
5 Identification of Management Targets in Notification Originators 61
6 Notification Filtering 62
7 Management Target Translation in Proxy Forwarder Applications 63
7.1 Management Target Translation for Request Forwarding 63
7.2 Management Target Translation for Notification Forwarding 64
8 Intellectual Property 65
9 Acknowledgments 66
10 Security Considerations 67
11 References 67
12 Editors' Addresses..... 69
A. Trap Configuration Example 70
B. Full Copyright Statement 72

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, as well as processing the response to a request 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 (4.5, 5.3, and 8) 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 [RFC2571], 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 [RFC2571].

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 is to be generated.

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

- The contextEngineID is provided by the command generator if it wishes to explicitly specify the location of the management information it is requesting.

- The contextName is provided by the command generator if it wishes to explicitly specify 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 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.

- 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.

(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 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. 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 (3) 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 (4) 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.

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.

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 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 [COEX].)

(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 originally received.

- The contextName is the value originally received.

- 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 [COEX].)

(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 received value.
- The securityModel is the received value.
- The securityName is the received value.
- The securityLevel is the received value.
- The contextEngineID is the received value.
- The contextName is the received value.
- The pduVersion is the received value.
- 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 received value.
- 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.

- If the original notification contains a Confirmed-Class PDU, then any outgoing management targets, for which the outgoing SNMP version does not support and PDU types which 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 [COEX].)

(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 a 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 originally received value.

- The securityModel is the originally received value.

- The securityName is the originally received value.

- The securityLevel is the originally received value.

- The contextEngineID is the originally received value.

- The contextName is the originally received value.

- 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 originally received value.

- 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 6.

- The use of the notification MIB for filtering notifications in notification originator applications is described in section 7.

- The use of the management target MIB and the proxy MIB in proxy forwarding applications is described in section 8.

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, [COEX] 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 (0x0B).

In addition, a tag value may not have a zero length. Generally, a particular MIB object may contain either

- a single tag value, in which case the value of the MIB object may not contain a delimiter character, or:
- a MIB object may contain 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 "9808040000Z"
    ORGANIZATION "IETF SNMPv3 Working Group"
    CONTACT-INFO
        "WG-email:   snmpv3@lists.tislabs.com
         Subscribe:  majordomo@lists.tislabs.com
                     In message body:  subscribe snmpv3

         Chair:      Russ Mundy
```

```
                    Trusted Information Systems
          Postal:   3060 Washington Rd
                    Glenwood MD 21738
                    USA
          EMail:    mundy@tislabs.com
          Phone:    +1-301-854-6889

          Co-editor: David B. Levi
                    SNMP Research, Inc.
          Postal:   3001 Kimberlin Heights Road
                    Knoxville, TN 37920-9716
          EMail:    levi@snmp.com
          Phone:    +1 423 573 1434

          Co-editor: Paul Meyer
                    Secure Computing Corporation
          Postal:   2675 Long Lake Road
                    Roseville, MN 55113
          EMail:    paul_meyer@securecomputing.com
          Phone:    +1 651 628 1592

          Co-editor: Bob Stewart
                    Cisco Systems, Inc.
          Postal:   170 West Tasman Drive
                    San Jose, CA 95134-1706
          EMail:    bstewart@cisco.com
          Phone:    +1 603 654 2686"
      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."
      REVISION    "9808040000Z"
      DESCRIPTION "Clarifications, published as
                   RFC2573."
      REVISION    "9707140000Z"
      DESCRIPTION "The initial revision, published as RFC2273."
      ::= { snmpModules 12 }

   snmpTargetObjects       OBJECT IDENTIFIER ::= { snmpTargetMIB 1 }
   snmpTargetConformance   OBJECT IDENTIFIER ::= { snmpTargetMIB 3 }

   SnmpTagValue ::= TEXTUAL-CONVENTION
       DISPLAY-HINT "255a"
       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 (0x0B).

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.

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 "255a"
      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 (0x0B).

           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 string

                - 'acme router'

                - 'host managerStation'

            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."
        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."
    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 sparsely augments the snmpTargetAddrTable with an object which is used to associate a set of filters with a particular management target.

The third table 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 "9808040000Z"
        ORGANIZATION "IETF SNMPv3 Working Group"
        CONTACT-INFO
            "WG-email:   snmpv3@lists.tislabs.com
             Subscribe:  majordomo@lists.tislabs.com
                         In message body:  subscribe snmpv3

             Chair:      Russ Mundy
                         Trusted Information Systems
             Postal:     3060 Washington Rd
                         Glenwood MD 21738
                         USA
             EMail:      mundy@tislabs.com
             Phone:      +1-301-854-6889

             Co-editor:  David B. Levi
                         SNMP Research, Inc.
             Postal:     3001 Kimberlin Heights Road
                         Knoxville, TN 37920-9716
             EMail:      levi@snmp.com
             Phone:      +1 423 573 1434

             Co-editor:  Paul Meyer
                         Secure Computing Corporation
             Postal:     2675 Long Lake Road
                         Roseville, MN 55113
             EMail:      paul_meyer@securecomputing.com
             Phone:      +1 651 628 1592

             Co-editor:  Bob Stewart
                         Cisco Systems, Inc.
             Postal:     170 West Tasman Drive
                         San Jose, CA 95134-1706
             EMail:      bstewart@cisco.com
             Phone:      +1 603 654 2686"
        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."
        REVISION    "9808040000Z"
        DESCRIPTION "Clarifications, published as
                     RFC2573"
        REVISION    "9707140000Z"
        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."
        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 of this conceptual 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 [RFC2573]."
    ::= { 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 [RFC2573]."
        DEFVAL { included }
        ::= { snmpNotifyFilterEntry 3 }

    snmpNotifyFilterStorageType OBJECT-TYPE
        SYNTAX      StorageType
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "The storage type of this conceptual 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 "9808040000Z"
    ORGANIZATION "IETF SNMPv3 Working Group"
    CONTACT-INFO
        "WG-email:   snmpv3@lists.tislabs.com
         Subscribe:  majordomo@lists.tislabs.com
                     In message body:  subscribe snmpv3

         Chair:      Russ Mundy
                     Trusted Information Systems
         Postal:     3060 Washington Rd
                     Glenwood MD 21738
                     USA
         EMail:      mundy@tislabs.com
         Phone:      +1-301-854-6889

         Co-editor:  David B. Levi
                     SNMP Research, Inc.
         Postal:     3001 Kimberlin Heights Road
                     Knoxville, TN 37920-9716
         EMail:      levi@snmp.com
         Phone:      +1 423 573 1434

         Co-editor:  Paul Meyer
                     Secure Computing Corporation
         Postal:     2675 Long Lake Road
                     Roseville, MN 55113
         EMail:      paul_meyer@securecomputing.com
         Phone:      +1 651 628 1592
```

```
          Co-editor:  Bob Stewart
                      Cisco Systems, Inc.
          Postal:     170 West Tasman Drive
                      San Jose, CA 95134-1706
          EMail:      bstewart@cisco.com
          Phone:      +1 603 654 2686"
      DESCRIPTION
          "This MIB module defines MIB objects which provide
           mechanisms to remotely configure the parameters
           used by a proxy forwarding application."
      REVISION    "9808040000Z"
```



```
      DESCRIPTION "Clarifications, published as
                   RFC2573."
      REVISION    "9707140000Z"
      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."
        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 each entry in the snmpNotifyTable to find the management targets to be used for generating notifications. Each active entry in this table identifies zero or more entries in the snmpTargetAddrTable. 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.

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 ([COEX] 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 lexicographically greatest instance of snmpNotifyFilterType among those which match 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, securityLevel, security model, and securityName 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, securityLevel, security model, and securityName 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 (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 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

[COEX] The SNMPv3 Working Group, Frye, R.,Levi, D., Wijnen, B., "Coexistence between Version 1, Version 2, and Version 3 of the Internet-standard Network Management Framework", Work in Progress.

[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.

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

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

[RFC1905] SNMPv2 Working Group, Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC1905, January 1996.

[RFC1907] SNMPv2 Working Group, Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Management Information Base for Version 2 of the Simple Network Management Protocol

(SNMPv2)", RFC1905, January 1996.

[RFC1908] SNMPv2 Working Group, Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Coexistence between Version 1 and Version 2 of the Internet-standard Network Management Framework", RFC1905, January 1996.

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

[RFC2571] Harrington, D., Presuhn, R. and B. Wijnen, "An Architecture for Describing SNMP Management Frameworks", RFC 2571, April 1999.

[RFC2572] Case, J., Harrington, D., Presuhn, R. and B. Wijnen, "Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)", RFC 2572, April 1999.

[RFC2575] Wijnen, B., Presuhn, R. and K. McCloghrie, "View-based Access Control Model for the Simple Network Management Protocol (SNMP)", RFC 2575, April 1999.

[RFC2573] Levi, D. B., Meyer, P. and B. Stewart, "SNMP Applications", RFC 2573, April 1999.

12. Editors' Addresses

David B. Levi
SNMP Research, Inc.
3001 Kimberlin Heights Road
Knoxville, TN 37920-9716
U.S.A.

Phone: +1 423 573 1434
EMail: levi@snmp.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
Cisco Systems, Inc.
170 West Tasman Drive

San Jose, CA 95134-1706
U.S.A.

Phone: +1 603 654 2686
EMail: bstewart@cisco.com

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, and two rows in the snmpTargetTable.

```
        snmpTargetAddrName          SnmpAdminString,
        snmpTargetAddrTDomain       TDomain,
        snmpTargetAddrTAddress      TAddress,
        snmpTargetAddrTimeout       TimeInterval,
        snmpTargetAddrRetryCount    Integer32,
        snmpTargetAddrTagList       SnmpAdminString,
        snmpTargetAddrParams        SnmpAdminString,
        snmpTargetAddrStorageType   StorageType,
        snmpTargetAddrRowStatus     RowStatus

      * 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     = 128.1.2.3/162
        snmpTargetAddrTagList      = "group2"
        snmpTargetAddrParams       = "AuthPriv-bob"
        snmpTargetAddrStorageType = readOnly(5)
        snmpTargetAddrRowStatus    = active(1)

      * snmpTargetParamsName                    = "AuthNoPriv-joe"
        snmpTargetParamsMPModel                 = 3m
```



```
        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
```


Appendix B. Full Copyright Statement

Copyright (C) The Internet Society (1999). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Acknowledgement

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

Network Working Group
Request for Comments: 2574
Obsoletes: 2274
Category: Standards Track

U. Blumenthal
IBM T. J. Watson Research
B. Wijnen
IBM T. J. Watson Research
April 1999

User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)

Status of this Memo

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

Copyright Notice

Copyright (C) The Internet Society (1999). All Rights Reserved.

Abstract

This document describes the User-based Security Model (USM) for SNMP version 3 for use in the SNMP architecture [RFC2571]. It defines the Elements of Procedure for providing SNMP message level security. This document also includes a MIB for remotely monitoring/managing the configuration parameters for this Security Model.

Table of Contents

1. Introduction 3
1.1. Threats 4
1.2. Goals and Constraints 5
1.3. Security Services 6
1.4. Module Organization 7
1.4.1. Timeliness Module 7
1.4.2. Authentication Protocol 8
1.4.3. Privacy Protocol 8
1.5. Protection against Message Replay, Delay and Redirection 8
1.5.1. Authoritative SNMP engine 8
1.5.2. Mechanisms 9
1.6. Abstract Service Interfaces 10
1.6.1. User-based Security Model Primitives for Authentication 11
1.6.2. User-based Security Model Primitives for Privacy 11
2. Elements of the Model 12
2.1. User-based Security Model Users 12

2.2. Replay Protection 13
2.2.1. msgAuthoritativeEngineID 13
2.2.2. msgAuthoritativeEngineBoots and msgAuthoritativeEngineTime 14
2.2.3. Time Window 15
2.3. Time Synchronization 15
2.4. SNMP Messages Using this Security Model 16
2.5. Services provided by the User-based Security Model 17
2.5.1. Services for Generating an Outgoing SNMP Message 17
2.5.2. Services for Processing an Incoming SNMP Message 19
2.6. Key Localization Algorithm. 21
3. Elements of Procedure 21
3.1. Generating an Outgoing SNMP Message 22
3.2. Processing an Incoming SNMP Message 25
4. Discovery 30
5. Definitions 31
6. HMAC-MD5-96 Authentication Protocol 50
6.1. Mechanisms 50
6.1.1. Digest Authentication Mechanism 50
6.2. Elements of the Digest Authentication Protocol 51
6.2.1. Users 51
6.2.2. msgAuthoritativeEngineID 51
6.2.3. SNMP Messages Using this Authentication Protocol 51
6.2.4. Services provided by the HMAC-MD5-96 Authentication Module 52
6.2.4.1. Services for Generating an Outgoing SNMP Message 52
6.2.4.2. Services for Processing an Incoming SNMP Message 53
6.3. Elements of Procedure 53
6.3.1. Processing an Outgoing Message 54
6.3.2. Processing an Incoming Message 54
7. HMAC-SHA-96 Authentication Protocol 55
7.1. Mechanisms 55
7.1.1. Digest Authentication Mechanism 56
7.2. Elements of the HMAC-SHA-96 Authentication Protocol 56
7.2.1. Users 56
7.2.2. msgAuthoritativeEngineID 57
7.2.3. SNMP Messages Using this Authentication Protocol 57
7.2.4. Services provided by the HMAC-SHA-96 Authentication Module 57
7.2.4.1. Services for Generating an Outgoing SNMP Message 57
7.2.4.2. Services for Processing an Incoming SNMP Message 58
7.3. Elements of Procedure 59
7.3.1. Processing an Outgoing Message 59
7.3.2. Processing an Incoming Message 60
8. CBC-DES Symmetric Encryption Protocol 61
8.1. Mechanisms 61
8.1.1. Symmetric Encryption Protocol 61
8.1.1.1. DES key and Initialization Vector. 62
8.1.1.2. Data Encryption. 63
8.1.1.3. Data Decryption 63
8.2. Elements of the DES Privacy Protocol 63

8.2.1. Users 63
8.2.2. msgAuthoritativeEngineID 64
8.2.3. SNMP Messages Using this Privacy Protocol 64
8.2.4. Services provided by the DES Privacy Module 64

8.2.4.1. Services for Encrypting Outgoing Data 64
8.2.4.2. Services for Decrypting Incoming Data 65
8.3. Elements of Procedure. 66
8.3.1. Processing an Outgoing Message 66
8.3.2. Processing an Incoming Message 66
9. Intellectual Property 67
10. Acknowledgements 67
11. Security Considerations 69
11.1. Recommended Practices 69
11.2. Defining Users 71
11.3. Conformance 72
11.4. Use of Reports 72
11.5. Access to the SNMP-USER-BASED-SM-MIB 72
12. References 73
13. Editors' Addresses 75
A.1. SNMP engine Installation Parameters 76
A.2. Password to Key Algorithm 78
A.2.1. Password to Key Sample Code for MD5 79
A.2.2. Password to Key Sample Code for SHA 80
A.3. Password to Key Sample Results 81
A.3.1. Password to Key Sample Results using MD5 81
A.3.2. Password to Key Sample Results using SHA 81
A.4. Sample encoding of msgSecurityParameters 82
A.5. Sample keyChange Results 83
A.5.1. Sample keyChange Results using MD5 83
A.5.2. Sample keyChange Results using SHA 84
B. Change Log 85
C. Full Copyright Statement 86

1. Introduction

The Architecture for describing Internet Management Frameworks [RFC2571] 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 [RFC2571].

This memo [RFC2274] 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 should prefer 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 [RFC2571]), 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 [RFC2571]), 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 [RFC1907]).

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.

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 [RFC2571]).

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 [RFC2572]).

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 usmUserTable that 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 Request 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-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 [RFC1906]) of an OCTET STRING formatted according to the UsmSecurityParameters defined in section 2.4, then the snmpInASNParseErrs counter [RFC1907] 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 "9901200000Z"            -- 20 Jan 1999, 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
                              Trusted Information Systems
                  postal:     3060 Washington Rd
                              Glenwood MD 21738
                              USA
                  email:      mundy@tislabs.com
                  phone:      +1-301-854-6889

                  Co-editor   Uri Blumenthal
```

```
                        IBM T. J. Watson Research
                postal:     30 Saw Mill River Pkwy,
                            Hawthorne, NY 10532
                            USA
                email:      uri@watson.ibm.com
                phone:      +1-914-784-7964

                Co-editor:  Bert Wijnen
                            IBM T. J. Watson Research
                postal:     Schagen 33
                            3461 GL Linschoten
                            Netherlands
                email:      wijnen@vnet.ibm.com
                phone:      +31-348-432-794
               "
   DESCRIPTION  "The management information definitions for the
                 SNMP User-based Security Model.
               "
--  Revision history

    REVISION     "9901200000Z"            -- 20 Jan 1999, midnight
    DESCRIPTION  "Clarifications, published as RFC2574"

    REVISION     "9711200000Z"            -- 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 responder 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
                 value as the corresponding secret of the clone-from
                 user.

                 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 [MD5], 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 [MD5] 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 [RFC2571]).

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 [RFC1906], 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 [MD5].

4) Prepend K2 to the result of the step 4 and calculate MD5 digest over it according to [MD5]. 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 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 [RFC2571]).

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 [RFC1906], 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 [RFC2571]).

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 [RFC1906] of an OCTET STRING representing the 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 [RFC1906] 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

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.

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 [RFC1905] 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 [RFC1907], 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 [RFC2575]).

12. References

[RFC1321] Rivest, R., "Message Digest Algorithm MD5", RFC 1321, April 1992.

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

[RFC1905] Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1905, January 1996.

[RFC1906] Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Transport Mappings for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1906, January 1996.

[RFC1907] Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Management Information Base for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1907 January 1996.

[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.

[RFC2571] Harrington, D., Presuhn, R. and B. Wijnen, "An Architecture for describing SNMP Management Frameworks", RFC 2571, April 1999.

[RFC2572] Case, J., Harrington, D., Presuhn, R. and B. Wijnen, "Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)", RFC 2572, April 1999.

[RF2575] Wijnen, B., Presuhn, R. and K. McCloghrie, "View-based Access Control Model for the Simple Network Management Protocol (SNMP)", RFC 2575, April 1999.

[Localized-Key] U. Blumenthal, N. C. Hien, B. Wijnen "Key Derivation for Network Management Applications" IEEE Network Magazine, April/May issue, 1997.

[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).

[DES-ANSI] Data Encryption Algorithm, American National Standards Institute. ANSI X3.92-1981, (December, 1980).

[DESO-NIST] DES Modes of Operation, National Institute of Standards and Technology. Federal Information Processing Standard (FIPS) Publication 81, (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).

[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)

13. Editors' Addresses

Uri Blumenthal
IBM T. J. Watson Research
30 Saw Mill River Pkwy,
Hawthorne, NY 10532
USA

Phone: +1-914-784-7064
EMail: uri@watson.ibm.com

Bert Wijnen
IBM T. J. Watson Research
Schagen 33
3461 GL Linschoten
Netherlands

Phone: +31-348-432-794
EMail: wijnen@vnet.ibm.com

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 [MD5]. 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 [MD5].

  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 the 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 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

C. Full Copyright Statement

Copyright (C) The Internet Society (1999). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Acknowledgement

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

Network Working Group
Request for Comments: 2576
Category: Standards Track

R. Frye
CoSine Communications
D. Levi
Nortel Networks
S. Routhier
Integrated Systems Inc.
B. Wijnen
Lucent Technologies
March 2000

Coexistence between Version 1, Version 2, and Version 3 of the Internet-standard Network Management Framework

Status of this Memo

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

Copyright Notice

Copyright (C) The Internet Society (2000). All Rights Reserved.

Abstract

The purpose of this document is to describe coexistence between version 3 of the Internet-standard Network Management Framework, (SNMPv3), version 2 of the Internet-standard Network Management Framework (SNMPv2), and the original Internet-standard Network Management Framework (SNMPv1). This document obsoletes RFC 1908 [13] and RFC2089 [14].

Table Of Contents

1 Overview 2
1.1 SNMPv1 3
1.2 SNMPv2 4
1.3 SNMPv3 4
1.4 SNMPv1 and SNMPv2 Access to MIB Data 5
2 SMI and Management Information Mappings 5
2.1 MIB Modules 6
2.1.1 Object Definitions 6
2.1.2 Trap and Notification Definitions 9
2.2 Compliance Statements 9
2.3 Capabilities Statements 10

3 Translating Notifications Parameters 10
3.1 Translating SNMPv1 Notification Parameters to SNMPv2 Notification Parameters 12
3.2 Translating SNMPv2 Notification Parameters to SNMPv1 Notification Parameters 13
4 Approaches to Coexistence in a Multi-lingual Network 14
4.1 Multi-lingual implementations 15
4.1.1 Command Generator 15
4.1.2 Command Responder 15
4.1.2.1 Handling Counter64 16
4.1.2.2 Mapping SNMPv2 Exceptions 16
4.1.2.2.1 Mapping noSuchObject and noSuchInstance 17
4.1.2.2.2 Mapping endOfMibView 17
4.1.2.3 Processing An SNMPv1 GetRequest 18
4.1.2.4 Processing An SNMPv1 GetNextRequest 19
4.1.2.5 Processing An SNMPv1 SetRequest 20
4.1.3 Notification Originator 20
4.1.4 Notification Receiver 21
4.2 Proxy Implementations 21
4.2.1 Upstream Version Greater Than Downstream Version 21
4.2.2 Upstream Version Less Than Downstream Version 22
4.3 Error Status Mappings 24
5 Message Processing Models and Security Models 25
5.1 Mappings 25
5.2 The SNMPv1 MP Model and SNMPv1 Community-based Security Model 26
5.2.1 Processing An Incoming Request 26
5.2.2 Generating An Outgoing Response 28
5.2.3 Generating An Outgoing Notification 28
5.3 The SNMP Community MIB Module 29
6 Intellectual Property 39
7 Acknowledgments 39
8 Security Considerations 40
9 References 40
10 Editor's Addresses 42
A. Changes From RFC1908 43
Full Copyright Statement 44

1. Overview

The purpose of this document is to describe coexistence between version 3 of the Internet-standard Network Management Framework, termed the SNMP version 3 framework (SNMPv3), version 2 of the Internet-standard Network Management Framework, termed the SNMP version 2 framework (SNMPv2), and the original Internet-standard Network Management Framework (SNMPv1).

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 [15].

There are four general aspects of coexistence described in this document. Each of these is described in a separate section:

- Conversion of MIB documents between SMIv1 and SMIv2 formats is documented in section 2.

- Mapping of notification parameters is documented in section 3.

- Approaches to coexistence between entities which support the various versions of SNMP in a multi-lingual network is documented in section 4. This section addresses the processing of protocol operations in multi-lingual implementations, as well as behaviour of proxy implementations.

- The SNMPv1 Message Processing Model and Community-Based Security Model, which provides mechanisms for adapting SNMPv1 into the View-Based Access Control Model (VACM) [20], is documented in section 5 (this section also addresses the SNMPv2c Message Processing Model and Community-Based Security Model).

1.1. SNMPv1

SNMPv1 is defined by these documents:

- STD 15, RFC 1157 [2] which defines the Simple Network Management Protocol (SNMPv1), the protocol used for network access to managed objects.

- STD 16, RFC 1155 [1] which defines the Structure of Management Information (SMIv1), the mechanisms used for describing and naming objects for the purpose of management.

- STD 16, RFC 1212 [3] which defines a more concise description mechanism, which is wholly consistent with the SMIv1.

- RFC 1215 [4] which defines a convention for defining Traps for use with the SMIv1.

Note that throughout this document, the term 'SMIv1' is used. This term generally refers to the information presented in RFC 1155, RFC 1212, and RFC 1215.

1.2. SNMPv2

SNMPv2 is defined by these documents:

- STD 58, RFC 2578 which defines Version 2 of the Structure of Management Information (SMIv2) [7].

- STD 58, RFC 2579 which defines common MIB "Textual Conventions"

[8].

- STD 58, RFC 2580 which defines Conformance Statements and requirements for defining agent and manager capabilities [9].

- RFC 1905 which defines the Protocol Operations used in processing [10].

- RFC 1906 which defines the Transport Mappings used "on the wire" [11].

- RFC 1907 which defines the basic Management Information Base for monitoring and controlling some basic common functions of SNMP entities [12].

Note that SMIv2 as used throughout this document refers to the first three documents listed above (RFCs 2578, 2579, and 2580).

The following document augments the definition of SNMPv2:

- RFC 1901 [6] is an Experimental definition for using SNMPv2 PDUs within a community-based message wrapper. This is referred to throughout this document as SNMPv2c.

1.3. SNMPv3

SNMPv3 is defined by these documents:

- RFC 2571 which defines an Architecture for Describing SNMP Management Frameworks [16].

- RFC 2572 which defines Message Processing and Dispatching [17].

- RFC 2573 which defines various SNMP Applications [18].

- RFC 2574 which defines the User-based Security Model (USM), providing for both Authenticated and Private (encrypted) SNMP messages [19].

- RFC 2575 which defines the View-based Access Control Model (VACM), providing the ability to limit access to different MIB objects on a per-user basis [20].

SNMPv3 also uses the SNMPv2 definitions of RFCs 1905 through 1907 and the SMIv2 definitions of 2578 through 2580 described above.

1.4. SNMPv1 and SNMPv2 Access to MIB Data

In several places, this document refers to 'SNMPv1 Access to MIB Data' and 'SNMPv2 Access to MIB Data'. These terms refer to the part of an SNMP agent which actually accesses instances of MIB objects,

and which actually initiates generation of notifications. Differences between the two types of access to MIB data are:

- Error-status values generated.
- Generation of exception codes.
- Use of the Counter64 data type.
- The format of parameters provided when a notification is generated.

SNMPv1 access to MIB data may generate SNMPv1 error-status values, will never generate exception codes nor use the Counter64 data type, and will provide SNMPv1 format parameters for generating notifications. Note also that SNMPv1 access to MIB data will actually never generate a readOnly error (a noSuchName error would always occur in the situation where one would expect a readOnly error).

SNMPv2 access to MIB data may generate SNMPv2 error-status values, may generate exception codes, may use the Counter64 data type, and will provide SNMPv2 format parameters for generating notifications. Note that SNMPv2 access to MIB data will never generate readOnly, noSuchName, or badValue errors.

Note that a particular multi-lingual implementation may choose to implement all access to MIB data as SNMPv2 access to MIB data, and perform the translations described herein for SNMPv1-based transactions.

2. SMI and Management Information Mappings

The SMIv2 approach towards describing collections of managed objects is nearly a proper superset of the approach defined in the SMIv1. For example, both approaches use an adapted subset of ASN.1 (1988)

[11] as the basis for a formal descriptive notation. Indeed, one might note that the SMIv2 approach largely codifies the existing practice for defining MIB modules, based on extensive experience with the SMIv1.

The following sections consider the three areas: MIB modules, compliance statements, and capabilities statements.

2.1. MIB Modules

MIB modules defined using the SMIv1 may continue to be used with protocol versions which use SNMPv2 PDUs. However, for the MIB modules to conform to the SMIv2, the following changes SHALL be made:

2.1.1. Object Definitions

In general, conversion of a MIB module does not require the deprecation of the objects contained therein. If the definition of an object is truly inadequate for its intended purpose, the object SHALL be deprecated or obsoleted, otherwise deprecation is not required.

(1) The IMPORTS statement MUST reference SNMPv2-SMI, instead of RFC1155-SMI and RFC-1212.

(2) The MODULE-IDENTITY macro MUST be invoked immediately after any IMPORTs statement.

(3) For any object with an integer-valued SYNTAX clause, in which the corresponding INTEGER does not have a range restriction (i.e., the INTEGER has neither a defined set of named-number enumerations nor an assignment of lower- and upper-bounds on its value), the object MUST have the value of its SYNTAX clause changed to Integer32, or have an appropriate range specified.

(4) For any object with a SYNTAX clause value of Counter, the object MUST have the value of its SYNTAX clause changed to Counter32.

(5) For any object with a SYNTAX clause value of Gauge, the object MUST have the value of its SYNTAX clause changed to Gauge32, or Unsigned32 where appropriate.

(6) For all objects, the ACCESS clause MUST be replaced by a MAX-ACCESS clause. The value of the MAX-ACCESS clause SHALL be the same as that of the ACCESS clause unless some other value makes "protocol sense" as the maximal level of access for the object. In particular, object types for which instances can be explicitly created by a protocol set operation, SHALL have a

MAX-ACCESS clause of "read-create". If the value of the ACCESS clause is "write-only", then the value of the MAX-ACCESS clause MUST be "read-write", and the DESCRIPTION clause SHALL note that reading this object will result in implementation-specific results. Note that in SMIv1, the ACCESS clause specifies the minimal required access, while in SMIv2, the MAX-ACCESS clause specifies the maximum allowed access. This should be considered when converting an ACCESS clause to a MAX-ACCESS clause.

(7) For all objects, if the value of the STATUS clause is "mandatory" or "optional", the value MUST be replaced with "current", "deprecated", or "obsolete" depending on the current usage of such objects.

(8) For any object not containing a DESCRIPTION clause, the object MUST have a DESCRIPTION clause defined.

(9) For any object corresponding to a conceptual row which does not have an INDEX clause, the object MUST have either an INDEX clause or an AUGMENTS clause defined.

(10) If any INDEX clause contains a reference to an object with a syntax of NetworkAddress, then a new object MUST be created and placed in this INDEX clause immediately preceding the object whose syntax is NetworkAddress. This new object MUST have a syntax of INTEGER, it MUST be not-accessible, and its value MUST always be 1. This approach allows one to convert a MIB module in SMIv1 format to one in SMIv2 format, and then use it with the SNMPv1 protocol with no impact to existing SNMPv1 agents and managers.

(11) For any object with a SYNTAX of NetworkAddress, the SYNTAX MUST be changed to IpAddress. Note that the use of NetworkAddress in new MIB documents is strongly discouraged (in fact, new MIB documents should be written using SMIv2, which does not define NetworkAddress).

(12) For any object containing a DEFVAL clause with an OBJECT IDENTIFIER value which is expressed as a collection of sub-identifiers, the value MUST be changed to reference a single ASN.1 identifier. This may require defining a series of new administrative assignments (OBJECT IDENTIFIERS) in order to define the single ASN.1 identifier.

(13) One or more OBJECT-GROUPS MUST be defined, and related objects SHOULD be collected into appropriate groups. Note that SMIv2 requires all OBJECT-TYPEs to be a member of at least one OBJECT-GROUP.

Other changes are desirable, but not necessary:

(1) Creation and deletion of conceptual rows is inconsistent using the SMIv1. The SMIv2 corrects this. As such, if the MIB module undergoes review early in its lifetime, and it contains conceptual tables which allow creation and deletion of conceptual rows, then the objects relating to those tables MAY be deprecated and replaced with objects defined using the new approach. The approach based on SMIv2 can be found in section 7 of RFC2578 [7], and the RowStatus and StorageType TEXTUAL-CONVENTIONs are described in section 2 of RFC2579 [8].

(2) For any object with a string-valued SYNTAX clause, in which the corresponding OCTET STRING does not have a size restriction (i.e., the OCTET STRING has no assignment of lower- and upper-bounds on its length), the bounds for the size of the object SHOULD be defined.

(3) All textual conventions informally defined in the MIB module SHOULD be redefined using the TEXTUAL-CONVENTION macro. Such a change would not necessitate deprecating objects previously defined using an informal textual convention.

(4) For any object which represents a measurement in some kind of

units, a UNITS clause SHOULD be added to the definition of that object.

(5) For any conceptual row which is an extension of another conceptual row, i.e., for which subordinate columnar objects both exist and are identified via the same semantics as the other conceptual row, an AUGMENTS clause SHOULD be used in place of the INDEX clause for the object corresponding to the conceptual row which is an extension.

Finally, to avoid common errors in SMIv1 MIB modules:

(1) For any non-columnar object that is instanced as if it were immediately subordinate to a conceptual row, the value of the STATUS clause of that object MUST be changed to "obsolete".

(2) For any conceptual row object that is not contained immediately subordinate to a conceptual table, the value of the STATUS clause of that object (and all subordinate objects) MUST be changed to "obsolete".

2.1.2. Trap and Notification Definitions

If a MIB module is changed to conform to the SMIv2, then each occurrence of the TRAP-TYPE macro MUST be changed to a corresponding invocation of the NOTIFICATION-TYPE macro:

(1) The IMPORTS statement MUST NOT reference RFC-1215 [4], and MUST reference SNMPv2-SMI instead.

(2) The ENTERPRISE clause MUST be removed.

(3) The VARIABLES clause MUST be renamed to the OBJECTS clause.

(4) A STATUS clause MUST be added, with an appropriate value. Normally the value should be 'current,' although 'deprecated' or 'obsolete' may be used as needed.

(5) The value of an invocation of the NOTIFICATION-TYPE macro is an OBJECT IDENTIFIER, not an INTEGER, and MUST be changed accordingly. Specifically, if the value of the ENTERPRISE clause is not 'snmp' then the value of the invocation SHALL be the value of the ENTERPRISE clause extended with two sub-identifiers, the first of which has the value 0, and the second has the value of the invocation of the TRAP-TYPE. If the value of the ENTERPRISE clause is 'snmp', then the value of the invocation of the NOTIFICATION-TYPE macro SHALL be mapped in the same manner as described in section 3.1 in this document.

(6) A DESCRIPTION clause MUST be added, if not already present.

(7) One or more NOTIFICATION-GROUPs MUST be defined, and related notifications MUST be collected into those groups. Note that SMIv2 requires that all NOTIFICATION-TYPEs be a member of at least one NOTIFICATION-GROUP.

2.2. Compliance Statements

For those information modules which are "standards track", a corresponding invocation of the MODULE-COMPLIANCE macro and related OBJECT-GROUP and/or NOTIFICATION-GROUP macros MUST be included within the information module (or in a companion information module), and any commentary text in the information module which relates to compliance SHOULD be removed. Typically this editing can occur when the information module undergoes review.

Note that a MODULE-COMPLIANCE statement is not required for a MIB document that is not on the standards track (for example, an enterprise MIB), though it may be useful in some circumstances to define a MODULE-COMPLIANCE statement for such a MIB document.

2.3. Capabilities Statements

RFC1303 [5] uses the MODULE-CONFORMANCE macro to describe an agent's capabilities with respect to one or more MIB modules. Converting such a description for use with the SMIv2 requires these changes:

(1) The macro name AGENT-CAPABILITIES SHOULD be used instead of MODULE-CONFORMANCE.

(2) The STATUS clause SHOULD be added, with a value of 'current'.

(3) All occurrences of the CREATION-REQUIRES clause MUST either be omitted if appropriate, or be changed such that the semantics are consistent with RFC2580 [9].

In order to ease coexistence, object groups defined in an SMIv1 compliant MIB module may be referenced by the INCLUDES clause of an invocation of the AGENT-CAPABILITIES macro: upon encountering a reference to an OBJECT IDENTIFIER subtree defined in an SMIv1 MIB module, all leaf objects which are subordinate to the subtree and have a STATUS clause value of mandatory are deemed to be INCLUDED. (Note that this method is ambiguous when different revisions of an SMIv1 MIB have different sets of mandatory objects under the same subtree; in such cases, the only solution is to rewrite the MIB using the SMIv2 in order to define the object groups unambiguously.)

3. Translating Notifications Parameters

This section describes how parameters used for generating notifications are translated between the format used for SNMPv1 notification protocol operations and the format used for SNMPv2 notification protocol operations. The parameters used to generate a notification are called 'notification parameters'. The format of parameters used for SNMPv1 notification protocol operations is refered to in this document as 'SNMPv1 notification parameters'. The format of parameters used for SNMPv2 notification protocol operations is refered to in this document as 'SNMPv2 notification parameters'.

The situations where notification parameters MUST be translated are:

- When an entity generates a set of notification parameters in a particular format, and the configuration of the entity indicates that the notification must be sent using an SNMP message version that requires the other format for notification parameters.

- When a proxy receives a notification that was sent using an SNMP message version that requires one format of notification parameters, and must forward the notification using an SNMP message version that requires the other format of notification parameters.

In addition, it MAY be desirable to translate notification parameters in a notification receiver application in order to present notifications to the end user in a consistent format.

Note that for the purposes of this section, the set of notification parameters is independent of whether the notification is to be sent as a trap or an inform.

SNMPv1 notification parameters consist of:

- An enterprise parameter (OBJECT IDENTIFIER).

- An agent-addr parameter (NetworkAddress).

- A generic-trap parameter (INTEGER).

- A specific-trap parameter (INTEGER).

- A time-stamp parameter (TimeTicks).

- A list of variable-bindings (VarBindList).

SNMPv2 notification parameters consist of:

- A sysUpTime parameter (TimeTicks). This appears in the first variable-binding in an SNMPv2-Trap-PDU or InformRequest-PDU.

- An snmpTrapOID parameter (OBJECT IDENTIFIER). This appears in the second variable-binding in an SNMPv2-Trap-PDU or InformRequest-PDU.

- A list of variable-bindings (VarBindList). This refers to all but the first two variable-bindings in an SNMPv2-Trap-PDU or InformRequest-PDU.

3.1. Translating SNMPv1 Notification Parameters to SNMPv2 Notification Parameters

The following procedure describes how to translate SNMPv1 notification parameters into SNMPv2 notification parameters:

(1) The SNMPv2 sysUpTime parameter SHALL be taken directly from the SNMPv1 time-stamp parameter.

(2) If the SNMPv1 generic-trap parameter is 'enterpriseSpecific(6)', the SNMPv2 snmpTrapOID parameter SHALL be the concatentation of the SNMPv1 enterprise parameter and two additional sub-identifiers, '0', and the SNMPv1 specific-trap parameter.

(3) If the SNMPv1 generic-trap parameter is not ' enterpriseSpecific(6)', the SNMPv2 snmpTrapOID parameter SHALL be the corresponding trap as defined in section 2 of RFC1907 [12]:

generic-trap parameter	snmpTrapOID.0
0	1.3.6.1.6.3.1.1.5.1 (coldStart)
1	1.3.6.1.6.3.1.1.5.2 (warmStart)
2	1.3.6.1.6.3.1.1.5.3 (linkDown)
3	1.3.6.1.6.3.1.1.5.4 (linkUp)
4	1.3.6.1.6.3.1.1.5.5 (authenticationFailure)
5	1.3.6.1.6.3.1.1.5.6 (egpNeighborLoss)

(4) The SNMPv2 variable-bindings SHALL be the SNMPv1 variable-bindings. In addition, if the translation is being performed by a proxy in order to forward a received trap, three additional variable-bindings will be appended, if these three additional variable-bindings do not already exist in the SNMPv1 variable-bindings. The name portion of the first additional variable binding SHALL contain snmpTrapAddress.0, and the value SHALL contain the SNMPv1 agent-addr parameter. The name portion of the second additional variable binding SHALL contain snmpTrapCommunity.0, and the value SHALL contain the value of the community-string field from the received SNMPv1 message

which contained the SNMPv1 Trap-PDU. The name portion of the third additional variable binding SHALL contain snmpTrapEnterprise.0 [12], and the value SHALL be the SNMPv1 enterprise parameter.

3.2. Translating SNMPv2 Notification Parameters to SNMPv1 Notification Parameters

The following procedure describes how to translate SNMPv2 notification parameters into SNMPv1 notification parameters:

(1) The SNMPv1 enterprise parameter SHALL be determined as follows:

- If the SNMPv2 snmpTrapOID parameter is one of the standard traps as defined in RFC1907 [12], then the SNMPv1 enterprise parameter SHALL be set to the value of the variable-binding in the SNMPv2 variable-bindings whose name is snmpTrapEnterprise.0 if that variable-binding exists. If it does not exist, the SNMPv1 enterprise parameter SHALL be set to the value ' snmpTraps' as defined in RFC1907 [12].

- If the SNMPv2 snmpTrapOID parameter is not one of the standard traps as defined in RFC1907 [12], then the SNMPv1 enterprise parameter SHALL be determined from the SNMPv2 snmpTrapOID parameter as follows:

  - If the next-to-last sub-identifier of the snmpTrapOID is zero, then the SNMPv1 enterprise SHALL be the SNMPv2 snmpTrapOID with the last 2 sub-identifiers removed, otherwise

  - If the next-to-last sub-identifier of the snmpTrapOID is non-zero, then the SNMPv1 enterprise SHALL be the SNMPv2 snmpTrapOID with the last sub-identifier removed.

(2) The SNMPv1 agent-addr parameter SHALL be determined based on the situation in which the translation occurs.

- If the translation occurs within a notification originator application, and the notification is to be sent over IP, the SNMPv1 agent-addr parameter SHALL be set to the IP address of the SNMP entity in which the notification originator resides. If the notification is to be sent over some other transport, the SNMPv1 agent-addr parameter SHALL be set to 0.0.0.0.

- If the translation occurs within a proxy application, the proxy must attempt to extract the original source of the notification from the variable-bindings. If the SNMPv2 variable-bindings contains a variable binding whose name is snmpTrapAddress.0,

the agent-addr parameter SHALL be set to the value of that variable binding. Otherwise, the SNMPv1 agent-addr parameter SHALL be set to 0.0.0.0.

(3) If the SNMPv2 snmpTrapOID parameter is one of the standard traps as defined in RFC1907 [12], the SNMPv1 generic-trap parameter SHALL be set as follows:

snmpTrapOID.0 parameter	generic-trap
1.3.6.1.6.3.1.1.5.1 (coldStart)	0
1.3.6.1.6.3.1.1.5.2 (warmStart)	1
1.3.6.1.6.3.1.1.5.3 (linkDown)	2
1.3.6.1.6.3.1.1.5.4 (linkUp)	3
1.3.6.1.6.3.1.1.5.5 (authenticationFailure)	4
1.3.6.1.6.3.1.1.5.6 (egpNeighborLoss)	5

Otherwise, the SNMPv1 generic-trap parameter SHALL be set to 6.

(4) If the SNMPv2 snmpTrapOID parameter is one of the standard traps as defined in RFC1907 [12], the SNMPv1 specific-trap parameter SHALL be set to zero. Otherwise, the SNMPv1 specific-trap parameter SHALL be set to the last sub-identifier of the SNMPv2 snmpTrapOID parameter.

(5) The SNMPv1 time-stamp parameter SHALL be taken directly from the SNMPv2 sysUpTime parameter.

(6) The SNMPv1 variable-bindings SHALL be the SNMPv2 variable-bindings. Note, however, that if the SNMPv2 variable-bindings contain any objects whose type is Counter64, the translation to SNMPv1 notification parameters cannot be performed. In this case, the notification cannot be encoded in an SNMPv1 packet (and so the notification cannot be sent using SNMPv1, see section 4.1.3 and section 4.2).

4. Approaches to Coexistence in a Multi-lingual Network

There are two basic approaches to coexistence in a multi-lingual network, multi-lingual implementations and proxy implementations. Multi-lingual implementations allow elements in a network to communicate with each other using an SNMP version which both elements support. This allows a multi-lingual implementation to communicate with any mono-lingual implementation, regardless of the SNMP version supported by the mono-lingual implementation.

Proxy implementations provide a mechanism for translating between SNMP versions using a third party network element. This allows network elements which support only a single, but different, SNMP version to communicate with each other. Proxy implementations are also useful for securing communications over an insecure link between two locally secure networks.

4.1. Multi-lingual implementations

This approach requires an entity to support multiple SNMP message versions. Typically this means supporting SNMPv1, SNMPv2c, and SNMPv3 message versions. The behaviour of various types of SNMP applications which support multiple message versions is described in the following sections. This approach allows entities which support multiple SNMP message versions to coexist with and communicate with entities which support only a single SNMP message version.

4.1.1. Command Generator

A command generator must select an appropriate message version when sending requests to another entity. One way to achieve this is to consult a local database to select the appropriate message version.

In addition, a command generator MUST 'downgrade' GetBulk requests to GetNext requests when selecting SNMPv1 as the message version for an outgoing request. This is done by simply changing the operation type to GetNext, ignoring any non-repeaters and max-repetitions values, and setting error-status and error-index to zero.

4.1.2. Command Responder

A command responder must be able to deal with both SNMPv1 and SNMPv2 access to MIB data. There are three aspects to dealing with this. A command responder must:

- Deal correctly with SNMPv2 access to MIB data that returns a Counter64 value while processing an SNMPv1 message,

- Deal correctly with SNMPv2 access to MIB data that returns one of the three exception values while processing an SNMPv1 message, and

- Map SNMPv2 error codes returned from SNMPv2 access to MIB data into SNMPv1 error codes when processing an SNMPv1 message.

Note that SNMPv1 error codes SHOULD NOT be used without any change when processing SNMPv2c or SNMPv3 messages, except in the case of proxy forwarding. In the case of proxy forwarding, for backwards compatibility, SNMPv1 error codes may be used without any change in a forwarded SNMPv2c or SNMPv3 message.

The following sections describe the behaviour of a command responder application which supports multiple SNMP message versions, and which uses some combination of SNMPv1 and SNMPv2 access to MIB data.

4.1.2.1. Handling Counter64

The SMIv2 [7] defines one new syntax that is incompatible with SMIv1. This syntax is Counter64. All other syntaxes defined by SMIv2 are compatible with SMIv1.

The impact on multi-lingual command responders is that they MUST NOT ever return a variable binding containing a Counter64 value in a response to a request that was received using the SNMPv1 message version.

Multi-lingual command responders SHALL take the approach that object instances whose type is Counter64 are implicitly excluded from view when processing an SNMPv1 message. So:

- On receipt of an SNMPv1 GetRequest-PDU containing a variable binding whose name field points to an object instance of type Counter64, a GetResponsePDU SHALL be returned, with an error-status of noSuchName and the error-index set to the variable binding that caused this error.

- On an SNMPv1 GetNextRequest-PDU, any object instance which contains a syntax of Counter64 SHALL be skipped, and the next accessible object instance that does not have the syntax of Counter64 SHALL be retrieved. If no such object instance exists, then an error-status of noSuchName SHALL be returned, and the error-index SHALL be set to the variable binding that caused this error.

- Any SNMPv1 request which contains a variable binding with a Counter64 value is ill-formed, so the foregoing rules do not apply. If that error is detected, a response SHALL NOT be returned, since it would contain a copy of the ill-formed variable binding. Instead, the offending PDU SHALL be discarded and the counter snmpInASNParseErrs SHALL be incremented.

4.1.2.2. Mapping SNMPv2 Exceptions

SNMPv2 provides a feature called exceptions, which allow an SNMPv2 Response PDU to return as much management information as possible, even when an error occurs. However, SNMPv1 does not support exceptions, and so an SNMPv1 Response PDU cannot return any management information, and can only return an error-status and error-index value.

When an SNMPv1 request is received, a command responder MUST check any variable bindings returned using SNMPv2 access to MIB data for exception values, and convert these exception values into SNMPv1 error codes.

The type of exception that can be returned when accessing MIB data and the action taken depends on the type of SNMP request.

- For a GetRequest, a noSuchObject or noSuchInstance exception may be returned.

- For a GetNextRequest, an endOfMibView exception may be returned.

- No exceptions will be returned for a SetRequest, and a GetBulkRequest should only be received in an SNMPv2c or SNMPv3 message, so these request types may be ignored when mapping exceptions.

Note that when a response contains multiple exceptions, it is an implementation choice as to which variable binding the error-index should reference.

4.1.2.2.1. Mapping noSuchObject and noSuchInstance

A noSuchObject or noSuchInstance exception generated by an SNMPv2 access to MIB data indicates that the requested object instance can not be returned. The SNMPv1 error code for this condition is noSuchName, and so the error-status field of the response PDU SHALL be set to noSuchName. Also, the error-index field SHALL be set to the index of the variable binding for which an exception occurred (there may be more than one and it is an implementation decision as to which is used), and the variable binding list from the original request SHALL be returned with the response PDU.

4.1.2.2.2. Mapping endOfMibView

When an SNMPv2 access to MIB data returns a variable binding containing an endOfMibView exception, it indicates that there are no object instances available which lexicographically follow the object in the request. In an SNMPv1 agent, this condition normally results in a noSuchName error, and so the error-status field of the response PDU SHALL be set to noSuchName. Also, the error-index field SHALL be set to the index of the variable binding for which an exception occurred (there may be more than one and it is an implementation decision as to which is used), and the variable binding list from the original request SHALL be returned with the response PDU.

4.1.2.3. Processing An SNMPv1 GetRequest

When processing an SNMPv1 GetRequest, the following procedures MUST be followed when using an SNMPv2 access to MIB data.

When such an access to MIB data returns response data using SNMPv2 syntax and error-status values, then:

(1) If the error-status is anything other than noError,

- The error status SHALL be translated to an SNMPv1 error-status using the table in section 4.3, "Error Status Mappings".

- The error-index SHALL be set to the position (in the original request) of the variable binding that caused the error-status.

- The variable binding list of the response PDU SHALL be made exactly the same as the variable binding list that was received in the original request.

(2) If the error-status is noError, the variable bindings SHALL be checked for any SNMPv2 exception (noSuchObject or noSuchInstance) or an SNMPv2 syntax that is unknown to SNMPv1 (Counter64). If there are any such variable bindings, one of those variable bindings SHALL be selected (it is an implementation choice as to which is selected), and:

- The error-status SHALL be set to noSuchName,

- The error-index SHALL be set to the position (in the variable binding list of the original request) of the selected variable binding, and

- The variable binding list of the response PDU SHALL be exactly the same as the variable binding list that was received in the original request.

(3) If there are no such variable bindings, then:

- The error-status SHALL be set to noError,

- The error-index SHALL be set to zero, and

- The variable binding list of the response SHALL be composed from the data as it is returned by the access to MIB data.

4.1.2.4. Processing An SNMPv1 GetNextRequest

When processing an SNMPv1 GetNextRequest, the following procedures MUST be followed when an SNMPv2 access to MIB data is called as part of processing the request. There may be repetitive accesses to MIB data to try to find the first object which lexicographically follows each of the objects in the request. This is implementation specific. These procedures are followed only for data returned when using

SNMPv2 access to MIB data. Data returned using SNMPv1 access to MIB data may be treated in the normal manner for an SNMPv1 request.

First, if the access to MIB data returns an error-status of anything other than noError:

(1) The error status SHALL be translated to an SNMPv1 error-status using the table in section 4.3, "Error Status Mappings".

(2) The error-index SHALL be set to the position (in the original request) of the variable binding that caused the error-status.

(3) The variable binding list of the response PDU SHALL be exactly the same as the variable binding list that was received in the original request.

Otherwise, if the access to MIB data returns an error-status of noError:

(1) Any variable bindings containing an SNMPv2 syntax of Counter64 SHALL be considered to be not in view, and MIB data SHALL be accessed as many times as is required until either a value other than Counter64 is returned, or an error occurs.

(2) If there is any variable binding that contains an SNMPv2 exception endOfMibView (there may be more than one, it is an implementation decision as to which is chosen):

- The error-status SHALL be set to noSuchName,

- The error-index SHALL be set to the position (in the variable binding list of the original request) of the variable binding that returned such an SNMPv2 exception, and

- The variable binding list of the response PDU SHALL be exactly the same as the variable binding list that was received in the original request.

(3) If there are no such variable bindings, then:

- The error-status SHALL be set to noError,

- The error-index SHALL be set to zero, and

- The variable binding list of the response SHALL be composed from the data as it is returned by the access to MIB data.

4.1.2.5. Processing An SNMPv1 SetRequest

When processing an SNMPv1 SetRequest, the following procedures MUST be followed when calling SNMPv2 MIB access routines.

When such MIB access routines return response data using SNMPv2 syntax and error-status values, and the error-status is anything other than noError, then:

- The error status SHALL be translated to an SNMPv1 error-status using the table in section 4.3, "Error Status Mappings".

- The error-index SHALL be set to the position (in the original request) of the variable binding that caused the error-status.

- The variable binding list of the response PDU SHALL be made exactly the same as the variable binding list that was received in the original request.

4.1.3. Notification Originator

A notification originator must be able to translate between SNMPv1 notifications parameters and SNMPv2 notification parameters in order to send a notification using a particular SNMP message version. If a notification is generated using SNMPv1 notification parameters, and configuration information specifies that notifications be sent using SNMPv2c or SNMPv3, the notification parameters must be translated to SNMPv2 notification parameters. Likewise, if a notification is generated using SNMPv2 notification parameters, and configuration information specifies that notifications be sent using SNMPv1, the notification parameters must be translated to SNMPv1 notification parameters. In this case, if the notification cannot be translated (due to the presence of a Counter64 type), it will not be sent using SNMPv1.

When a notification originator generates a notification, using parameters obtained from the SNMP-TARGET-MIB and SNMP-NOTIFICATION-MIB, if the SNMP version used to generate the notification is SNMPv1, the PDU type used will always be a TrapPDU, regardless of whether the value of snmpNotifyType is trap(1) or inform(2).

Note also that access control and notification filtering are performed in the usual manner for notifications, regardless of the SNMP message version to be used when sending a notification. The parameters for performing access control are found in the usual manner (i.e., from inspecting the SNMP-TARGET-MIB and SNMP-NOTIFICATION-MIB). In particular, when generating an SNMPv1 Trap, in order to perform the access check specified in [18], section 3.3, bullet (3), the notification originator may need to generate a value for snmpTrapOID.0 as described in section 3.1, bullets (2) and (3) of this document. If the SNMPv1 notification parameters being used were previously translated from a set of SNMPv2 notification parameters, this value may already be known, in which case it need not be generated.

4.1.4. Notification Receiver

There are no special requirements of a notification receiver. However, an implementation may find it useful to allow a higher level application to request whether notifications should be delivered to a higher level application using SNMPv1 notification parameter or SNMPv2 notification parameters. The notification receiver would then translate notification parameters when required in order to present a notification using the desired set of parameters.

4.2. Proxy Implementations

A proxy implementation may be used to enable communication between entities which support different SNMP message versions. This is accomplished in a proxy forwarder application by performing translations on PDUs. These translations depend on the PDU type, the SNMP version of the packet containing a received PDU, and the SNMP version to be used to forward a received PDU. The following sections describe these translations. In all cases other than those described below, the proxy SHALL forward a received PDU without change, subject to size constraints as defined in section 5.3 (Community MIB) of this document. Note that in the following sections, the 'Upstream Version' refers to the version used between the command generator and the proxy, and the 'Downstream Version' refers to the version used between the proxy and the command responder, regardless of the PDU type or direction.

4.2.1. Upstream Version Greater Than Downstream Version

- If a GetBulkRequest-PDU is received and must be forwarded using the SNMPv1 message version, the proxy forwarder SHALL set the non-repeaters and max-repetitions fields to 0, and SHALL set the tag of the PDU to GetNextRequest-PDU.

- If a GetResponse-PDU is received whose error-status field has a value of 'tooBig', the message will be forwarded using the SNMPv2c or SNMPv3 message version, and the original request received by the proxy was not a GetBulkRequest-PDU, the proxy forwarder SHALL remove the contents of the variable-bindings field before forwarding the response.

- If a GetResponse-PDU is received whose error-status field has a value of 'tooBig,' and the message will be forwarded using the SNMPv2c or SNMPv3 message version, and the original request received by the proxy was a GetBulkRequest-PDU, the proxy forwarder SHALL re-send the forwarded request (which would have been altered to be a GetNextRequest-PDU) with all but the first variable-binding removed. The proxy forwarder SHALL only re-send such a request a single time. If the resulting GetResponse-PDU also contains an error-status field with a value of 'tooBig,' then the proxy forwarder SHALL remove the contents of the variable-bindings field, and change the error-status field to 'noError' before forwarding the response. Note that if the original request only contained a single variable-binding, the proxy may skip re-

sending the request and simply remove the variable-bindings and change the error-status to 'noError.'

- If a Trap-PDU is received, and will be forwarded using the SNMPv2c or SNMPv3 message version, the proxy SHALL apply the translation rules described in section 3, and SHALL forward the notification as an SNMPv2-Trap-PDU.

  Note that when an SNMPv1 agent generates a message containing a Trap-PDU which is subsequently forwarded by one or more proxy forwarders using SNMP versions other than SNMPv1, the community string and agent-addr fields from the original message generated by the SNMPv1 agent will be preserved through the use of the snmpTrapAddress and snmpTrapCommunity nobjects.

4.2.2. Upstream Version Less Than Downstream Version

- If a GetResponse-PDU is received in response to a GetRequest-PDU (previously generated by the proxy) which contains variable-bindings of type Counter64 or which contain an SNMPv2 exception code, and the message would be forwarded using the SNMPv1 message version, the proxy MUST generate an alternate response PDU consisting of the request-id and variable bindings from the original SNMPv1 request, containing a noSuchName error-status value, and containing an error-index value indicating the position of the variable-binding containing the Counter64 type or exception code.

- If a GetResponse-PDU is received in response to a GetNextRequest-PDU (previously generated by the proxy) which contains variable-bindings that contain an SNMPv2 exception code, and the message would be forwarded using the SNMPv1 message version, the proxy MUST generate an alternate response PDU consisting of the request-id and variable bindings from the original SNMPv1 request, containing a noSuchName error-status value, and containing an error-index value indicating the position of the variable-binding containing the exception code.

- If a GetResponse-PDU is received in response to a GetNextRequest-PDU (previously generated by the proxy) which contains variable-bindings of type Counter64, the proxy MUST re-send the entire GetNextRequest-PDU, with the following modifications. For any variable bindings in the received GetResponse which contained Counter64 types, the proxy substitutes the object names of these variable bindings for the corresponding object names in the previously-sent GetNextRequest. The proxy MUST repeat this process until no Counter64 objects are returned. Note that an implementation may attempt to optimize this process of skipping Counter64 objects. One approach to such an optimization would be to replace the last sub-identifier of the object names of varbinds containing a Counter64 type with 65535 if that sub-identifier is less than 65535, or with 4294967295 if that sub-identifier is

greater than 65535. This approach should skip multiple instances of the same Counter64 object, while maintaining compatibility with some broken agent implementations (which only use 16-bit integers for sub-identifiers).

Deployment Hint: The process of repeated GetNext requests used by a proxy when Counter64 types are returned can be expensive. When deploying a proxy, this can be avoided by configuring the target agents to which the proxy forwards requests in a manner such that any objects of type Counter64 are in fact not-in-view for the principal that the proxy is using when communicating with these agents.

- If a GetResponse-PDU is received which contains an SNMPv2 error-status value of wrongValue, wrongEncoding, wrongType, wrongLength, inconsistentValue, noAccess, notWritable, noCreation, inconsistentName, resourceUnavailable, commitFailed, undoFailed, or authorizationError, the error-status value is modified using the mappings in section 4.3.

- If an SNMPv2-Trap-PDU is received, and will be forwarded using the SNMPv1 message version, the proxy SHALL apply the translation rules described in section 3, and SHALL forward the notification

as a Trap-PDU. Note that if the translation fails due to the existence of a Counter64 data-type in the received SNMPv2-Trap-PDU, the trap cannot be forwarded using SNMPv1.

- If an InformRequest-PDU is received, any configuration information indicating that it would be forwarded using the SNMPv1 message version SHALL be ignored. An InformRequest-PDU can only be forwarded using the SNMPv2c or SNMPv3 message version. The InformRequest-PDU may still be forwarded if there is other configuration information indicating that it should be forwarded using SNMPv2c or SNMPv3.

4.3. Error Status Mappings

The following tables shows the mappings of SNMPv1 error-status values into SNMPv2 error-status values, and the mappings of SNMPv2 error-status values into SNMPv1 error-status values.

SNMPv1 error-status	SNMPv2 error-status
noError	noError
tooBig	tooBig
noSuchName	noSuchName
badValue	badValue
genErr	genErr

SNMPv2 error-status	SNMPv1 error-status

noError	noError
tooBig	tooBig
genErr	genErr
wrongValue	badValue
wrongEncoding	badValue
wrongType	badValue
wrongLength	badValue
inconsistentValue	badValue
noAccess	noSuchName
notWritable	noSuchName
noCreation	noSuchName
inconsistentName	noSuchName
resourceUnavailable	genErr
commitFailed	genErr
undoFailed	genErr
authorizationError	noSuchName

Whenever the SNMPv2 error-status value of authorizationError is translated to an SNMPv1 error-status value of noSuchName, the value of snmpInBadCommunityUses MUST be incremented.

5. Message Processing Models and Security Models

In order to adapt SNMPv1 (and SNMPv2c) into the SNMP architecture, the following models are defined in this document:

- The SNMPv1 Message Processing Model
- The SNMPv1 Community-Based Security Model

The following models are also described in this document:

- The SNMPv2c Message Processing Model
- The SNMPv2c Community-Based Security Model

In most respects, the SNMPv1 Message Processing Model and the SNMPv2c Message Processing Model are identical, and so these are not discussed independently in this document. Differences between the two models are described as required.

Similarly, the SNMPv1 Community-Based Security Model and the SNMPv2c Community-Based Security Model are nearly identical, and so are not discussed independently. Differences between these two models are also described as required.

5.1. Mappings

The SNMPv1 (and SNMPv2c) Message Processing Model and Security Model

require mappings between parameters used in SNMPv1 (and SNMPv2c) messages, and the version independent parameters used in the SNMP architecture [16]. The parameters which MUST be mapped consist of the SNMPv1 (and SNMPv2c) community name, and the SNMP securityName and contextEngineID/contextName pair. A MIB module (the SNMP-COMMUNITY-MIB) is provided in this document in order to perform these mappings. This MIB provides mappings in both directions, that is, a community name may be mapped to a securityName, contextEngineID, and contextName, or the combination of securityName, contextEngineID, and contextName may be mapped to a community name.

5.2. The SNMPv1 MP Model and SNMPv1 Community-based Security Model

The SNMPv1 Message Processing Model handles processing of SNMPv1 messages. The processing of messages is handled generally in the same manner as described in RFC1157 [2], with differences and clarifications as described in the following sections. The SnmpMessageProcessingModel value for SNMPv1 is 0 (the value for SNMPv2c is 1).

5.2.1. Processing An Incoming Request

In RFC1157 [2], section 4.1, item (3) for an entity which receives a message, states that various parameters are passed to the 'desired authentication scheme.' The desired authentication scheme in this case is the SNMPv1 Community-Based Security Model, which will be called using the processIncomingMsg ASI. The parameters passed to this ASI are:

- The messageProcessingModel, which will be 0 (or 1 for SNMPv2c).

- The maxMessageSize, which should be the maximum size of a message that the receiving entity can generate (since there is no such value in the received message).

- The securityParameters, which consist of the community string and the message's source and destination transport domains and addresses.

- The securityModel, which will be 1 (or 2 for SNMPv2c).

- The securityLevel, which will be noAuthNoPriv.

- The wholeMsg and wholeMsgLength.

The Community-Based Security Model will attempt to select a row in the snmpCommunityTable. This is done by performing a search through

the snmpCommunityTable in lexicographic order. The first entry for which the following matching criteria are satisfied will be selected:

- The community string is equal to the snmpCommunityName value.

- If the snmpCommunityTransportTag is an empty string, it is ignored for the purpose of matching. If the snmpCommunityTransportTag is not an empty string, the transportDomain and transportAddress from which the message was received must match one of the entries in the snmpTargetAddrTable selected by the snmpCommunityTransportTag

value. The snmpTargetAddrTMask object is used as described in section 5.3 when checking whether the transportDomain and transportAddress matches a entry in the snmpTargetAddrTable.

If no such entry can be found, an authentication failure occurs as described in RFC1157 [2], and the snmpInBadCommunityNames counter is incremented.

The parameters returned from the Community-Based Security Model are:

- The securityEngineID, which will always be the local value of snmpEngineID.0.

- The securityName.

- The scopedPDU. Note that this parameter will actually consist of three values, the contextSnmpEngineID, the contextName, and the PDU. These must be separate values, since the first two do not actually appear in the message.

- The maxSizeResponseScopedPDU.

- The securityStateReference.

The appropriate SNMP application will then be called (depending on the value of the contextEngineID and the request type in the PDU) using the processPdu ASI. The parameters passed to this ASI are:

- The messageProcessingModel, which will be 0 (or 1 for SNMPv2c).

- The securityModel, which will be 1 (or 2 for SNMPv2c).

- The securityName, which was returned from the call to processIncomingMsg.

- The securityLevel, which is noAuthNoPriv.

- The contextEngineID, which was returned as part of the ScopedPDU from the call to processIncomingMsg.

- The contextName, which was returned as part of the ScopedPDU from the call to processIncomingMsg.

- The pduVersion, which should indicate an SNMPv1 version PDU (if the message version was SNMPv2c, this would be an SNMPv2 version PDU).

- The PDU, which was returned as part of the ScopedPDU from the call to processIncomingMsg.

- The maxSizeResponseScopedPDU which was returned from the call to processIncomingMsg.

- The stateReference which was returned from the call to processIncomingMsg.

The SNMP application should process the request as described previously in this document. Note that access control is applied by an SNMPv3 command responder application as usual. The parameters as passed to the processPdu ASI will be used in calls to the isAccessAllowed ASI.

5.2.2. Generating An Outgoing Response

There is no special processing required for generating an outgoing response. However, the community string used in an outgoing response must be the same as the community string from the original request. The original community string MUST be present in the stateReference information of the original request.

5.2.3. Generating An Outgoing Notification

In a multi-lingual SNMP entity, the parameters used for generating notifications will be obtained by examining the SNMP-TARGET-MIB and SNMP-NOTIFICATION-MIB. These parameters will be passed to the SNMPv1 Message Processing Model using the sendPdu ASI. The SNMPv1 Message Processing Model will attempt to locate an appropriate community string in the snmpCommunityTable based on the parameters passed to the sendPdu ASI. This is done by performing a search through the snmpCommunityTable in lexicographic order. The first entry for which the following matching criteria are satisfied will be selected:

- The securityName must be equal to the snmpCommunitySecurityName value.

- The contextEngineID must be equal to the snmpCommunityContextEngineID value.

- The contextName must be equal to the snmpCommunityContextName value.

- If the snmpCommunityTransportTag is an empty string, it is ignored for the purpose of matching. If the snmpCommunityTransportTag is not an empty string, the

transportDomain and transportAddress must match one of the entries in the snmpTargetAddrTable selected by the snmpCommunityTransportTag value.

If no such entry can be found, the notification is not sent. Otherwise, the community string used in the outgoing notification will be the value of the snmpCommunityName column of the selected row.

5.3. The SNMP Community MIB Module

The SNMP-COMMUNITY-MIB contains objects for mapping between community strings and version-independent SNMP message parameters. In addition, this MIB provides a mechanism for performing source address validation on incoming requests, and for selecting community strings based on target addresses for outgoing notifications. These two features are accomplished by providing a tag in the snmpCommunityTable which selects sets of entries in the snmpTargetAddrTable [18]. In addition, the SNMP-COMMUNITY-MIB augments the snmpTargetAddrTable with a transport address mask value and a maximum message size value. These values are used only where explicitly stated. In cases where the snmpTargetAddrTable is used without mention of these augmenting values, the augmenting values should be ignored.

The mask value, snmpTargetAddrTMask, allows selected entries in the snmpTargetAddrTable to specify multiple addresses (rather than just a single address per entry). This would typically be used to specify a subnet in an snmpTargetAddrTable rather than just a single address. The mask value is used to select which bits of a transport address must match bits of the corresponding instance of snmpTargetAddrTAddress, in order for the transport address to match a particular entry in the snmpTargetAddrTable. The value of an instance of snmpTargetAddrTMask must always be an OCTET STRING whose length is either zero or the same as that of the corresponding instance of snmpTargetAddrTAddress.

Note that the snmpTargetAddrTMask object is only used where explicitly stated. In particular, it is not used when generating notifications (i.e., when generating notifications, entries in the snmpTargetAddrTable only specify individual addresses).

When checking whether a transport address matches an entry in the snmpTargetAddrTable, if the value of snmpTargetAddrTMask is a zero-length OCTET STRING, the mask value is ignored, and the value of snmpTargetAddrTAddress must exactly match a transport address. Otherwise, each bit of each octet in the snmpTargetAddrTMask value corresponds to the same bit of the same octet in the

snmpTargetAddrTAddress value. For bits that are set in the snmpTargetAddrTMask value (i.e., bits equal to 1), the corresponding bits in the snmpTargetAddrTAddress value must match the bits in a transport address. If all such bits match, the transport address is matched by that snmpTargetAddrTable entry. Otherwise, the transport address is not matched.

The maximum message size value, snmpTargetAddrMMS, is used to determine the maximum message size acceptable to another SNMP entity when the value cannot be determined from the protocol.

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

IMPORTS
    IpAddress,
    MODULE-IDENTITY,
    OBJECT-TYPE,
    Integer32,
    snmpModules
        FROM SNMPv2-SMI
    RowStatus,
    StorageType
        FROM SNMPv2-TC
    SnmpAdminString,
    SnmpEngineID
        FROM SNMP-FRAMEWORK-MIB
    SnmpTagValue,
    snmpTargetAddrEntry
        FROM SNMP-TARGET-MIB
    MODULE-COMPLIANCE,
    OBJECT-GROUP
        FROM SNMPv2-CONF;

snmpCommunityMIB MODULE-IDENTITY
    LAST-UPDATED "200003060000Z"          -- 6 Mar 2000, 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
                              TIS Labs at Network Associates
                  Postal:     3060 Washington Rd
                              Glenwood MD 21738
                              USA
                  Email:      mundy@tislabs.com
                  Phone:      +1-301-854-6889
```

```
                Co-editor:  Rob Frye
                            CoSine Communications
                Postal:     1200 Bridge Parkway
                            Redwood City, CA 94065
                            USA
                E-mail:     rfrye@cosinecom.com
                Phone:      +1 703 725 1130

                Co-editor:  David B. Levi
                            Nortel Networks
                Postal:     3505 Kesterwood Drive
                            Knoxville, TN 37918
                E-mail:     dlevi@nortelnetworks.com
                Phone:      +1 423 686 0432

                Co-editor:  Shawn A. Routhier
                            Integrated Systems Inc.
                Postal:     333 North Ave 4th Floor
                            Wakefield, MA 01880
                E-mail:     sar@epilogue.com
                Phone:      +1 781 245 0804

                Co-editor:  Bert Wijnen
                            Lucent Technologies
                Postal:     Schagen 33
                            3461 GL Linschoten
                            Netherlands
                Email:      bwijnen@lucent.com
                Phone:      +31-348-407-775
               "
```


```
        DESCRIPTION
            "This MIB module defines objects to help support coexistence
             between SNMPv1, SNMPv2c, and SNMPv3."
        REVISION "200003060000Z" -- 6 Mar 2000
        DESCRIPTION "This version published as RFC 2576."
        REVISION "199905130000Z" -- 13 May 1999
        DESCRIPTION "The Initial Revision"
    ::= { snmpModules 18 }

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

snmpCommunityMIBObjects     OBJECT IDENTIFIER ::= { snmpCommunityMIB 1 }
snmpCommunityMIBConformance OBJECT IDENTIFIER ::= { snmpCommunityMIB 2 }

--
-- The snmpCommunityTable contains a database of community strings.
-- This table provides mappings between community strings, and the
```

```
-- parameters required for View-based Access Control.
--

snmpCommunityTable OBJECT-TYPE
    SYNTAX       SEQUENCE OF SnmpCommunityEntry
    MAX-ACCESS   not-accessible
    STATUS       current
    DESCRIPTION
        "The table of community strings configured in the SNMP
         engine's Local Configuration Datastore (LCD)."
    ::= { snmpCommunityMIBObjects 1 }

snmpCommunityEntry OBJECT-TYPE
    SYNTAX       SnmpCommunityEntry
    MAX-ACCESS   not-accessible
    STATUS       current
    DESCRIPTION
        "Information about a particular community string."
    INDEX       { IMPLIED snmpCommunityIndex }
    ::= { snmpCommunityTable 1 }

SnmpCommunityEntry ::= SEQUENCE {
    snmpCommunityIndex                SnmpAdminString,
    snmpCommunityName                 OCTET STRING,
    snmpCommunitySecurityName         SnmpAdminString,
    snmpCommunityContextEngineID      SnmpEngineID,
    snmpCommunityContextName          SnmpAdminString,
    snmpCommunityTransportTag         SnmpTagValue,
    snmpCommunityStorageType          StorageType,
    snmpCommunityStatus               RowStatus
}

snmpCommunityIndex OBJECT-TYPE
    SYNTAX      SnmpAdminString (SIZE(1..32))
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "The unique index value of a row in this table."
    ::= { snmpCommunityEntry 1 }

snmpCommunityName OBJECT-TYPE
    SYNTAX       OCTET STRING
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION
        "The community string for which a row in this table
         represents a configuration."
    ::= { snmpCommunityEntry 2 }
```



```
snmpCommunitySecurityName OBJECT-TYPE
    SYNTAX       SnmpAdminString (SIZE(1..32))
    MAX-ACCESS   read-create
    STATUS       current
```

```
    DESCRIPTION
        "A human readable string representing the corresponding
         value of snmpCommunityName in a Security Model
         independent format."
    ::= { snmpCommunityEntry 3 }

snmpCommunityContextEngineID OBJECT-TYPE
    SYNTAX       SnmpEngineID
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION
        "The contextEngineID indicating the location of the
         context in which management information is accessed
         when using the community string specified by the
         corresponding instance of snmpCommunityName.

         The default value is the snmpEngineID of the entity in
         which this object is instantiated."
    ::= { snmpCommunityEntry 4 }

snmpCommunityContextName OBJECT-TYPE
    SYNTAX       SnmpAdminString (SIZE(0..32))
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION
        "The context in which management information is accessed
         when using the community string specified by the corresponding
         instance of snmpCommunityName."
    DEFVAL      { ''H }   -- the empty string
    ::= { snmpCommunityEntry 5 }

snmpCommunityTransportTag OBJECT-TYPE
    SYNTAX       SnmpTagValue
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION
        "This object specifies a set of transport endpoints
         from which a command responder application will accept
         management requests.  If a management request containing
         this community is received on a transport endpoint other
         than the transport endpoints identified by this object,
         the request is deemed unauthentic.

         The transports identified by this object are specified
```



```
         in the snmpTargetAddrTable.  Entries in that table
         whose snmpTargetAddrTagList contains this tag value
         are identified.

         If the value of this object has zero-length, transport
         endpoints are not checked when authenticating messages
         containing this community string."
    DEFVAL      { ''H }   -- the empty string
```

```
    ::= { snmpCommunityEntry 6 }

snmpCommunityStorageType OBJECT-TYPE
    SYNTAX        StorageType
    MAX-ACCESS    read-create
    STATUS        current
    DESCRIPTION
        "The storage type for this conceptual row in the
         snmpCommunityTable.  Conceptual rows having the value
         'permanent' need not allow write-access to any
         columnar object in the row."
    ::= { snmpCommunityEntry 7 }

snmpCommunityStatus OBJECT-TYPE
    SYNTAX        RowStatus
    MAX-ACCESS    read-create
    STATUS        current
    DESCRIPTION
        "The status of this conceptual row in the snmpCommunityTable.

         An entry in this table is not qualified for activation
         until instances of all corresponding columns have been
         initialized, either through default values, or through
         Set operations.  The snmpCommunityName and
         snmpCommunitySecurityName objects must be explicitly set.

         There is no restriction on setting columns in this table
         when the value of snmpCommunityStatus is active(1)."
    ::= { snmpCommunityEntry 8 }

--
-- The snmpTargetAddrExtTable
--

snmpTargetAddrExtTable OBJECT-TYPE
    SYNTAX        SEQUENCE OF SnmpTargetAddrExtEntry
    MAX-ACCESS    not-accessible
    STATUS        current
    DESCRIPTION
        "The table of mask and mms values associated with the
```



```
         snmpTargetAddrTable.

         The snmpTargetAddrExtTable augments the
         snmpTargetAddrTable with a transport address mask value
         and a maximum message size value.  The transport address
         mask allows entries in the snmpTargetAddrTable to define
         a set of addresses instead of just a single address.
         The maximum message size value allows the maximum
         message size of another SNMP entity to be configured for
         use in SNMPv1 (and SNMPv2c) transactions, where the
         message format does not specify a maximum message size."
    ::= { snmpCommunityMIBObjects 2 }
```

```
snmpTargetAddrExtEntry OBJECT-TYPE
    SYNTAX       SnmpTargetAddrExtEntry
    MAX-ACCESS   not-accessible
    STATUS       current
    DESCRIPTION
        "Information about a particular mask and mms value."
    AUGMENTS       { snmpTargetAddrEntry }
    ::= { snmpTargetAddrExtTable 1 }

SnmpTargetAddrExtEntry ::= SEQUENCE {
    snmpTargetAddrTMask              OCTET STRING,
    snmpTargetAddrMMS                Integer32
}

snmpTargetAddrTMask OBJECT-TYPE
    SYNTAX      OCTET STRING (SIZE (0..255))
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "The mask value associated with an entry in the
         snmpTargetAddrTable.  The value of this object must
         have the same length as the corresponding instance of
         snmpTargetAddrTAddress, or must have length 0.  An
         attempt to set it to any other value will result in
         an inconsistentValue error.

         The value of this object allows an entry in the
         snmpTargetAddrTable to specify multiple addresses.
         The mask value is used to select which bits of
         a transport address must match bits of the corresponding
         instance of snmpTargetAddrTAddress, in order for the
         transport address to match a particular entry in the
         snmpTargetAddrTable.  Bits which are 1 in the mask
         value indicate bits in the transport address which
         must match bits in the snmpTargetAddrTAddress value.
```



```
         Bits which are 0 in the mask indicate bits in the
         transport address which need not match.  If the
         length of the mask is 0, the mask should be treated
         as if all its bits were 1 and its length were equal
         to the length of the corresponding value of
         snmpTargetAddrTable.

         This object may not be modified while the value of the
         corresponding instance of snmpTargetAddrRowStatus is
         active(1).  An attempt to set this object in this case
         will result in an inconsistentValue error."
    DEFVAL { ''H }
    ::= { snmpTargetAddrExtEntry 1 }

snmpTargetAddrMMS OBJECT-TYPE
    SYNTAX      Integer32 (0|484..2147483647)
```

```
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "The maximum message size value associated with an entry
         in the snmpTargetAddrTable."
    DEFVAL { 484 }
    ::= { snmpTargetAddrExtEntry 2 }

--
-- The snmpTrapAddress and snmpTrapCommunity objects are included
-- in notifications that are forwarded by a proxy, which were
-- originally received as SNMPv1 Trap messages.
--

snmpTrapAddress OBJECT-TYPE
    SYNTAX      IpAddress
    MAX-ACCESS  accessible-for-notify
    STATUS      current
    DESCRIPTION
        "The value of the agent-addr field of a Trap PDU which
         is forwarded by a proxy forwarder application using
         an SNMP version other than SNMPv1.  The value of this
         object SHOULD contain the value of the agent-addr field
         from the original Trap PDU as generated by an SNMPv1
         agent."
    ::= { snmpCommunityMIBObjects 3 }

snmpTrapCommunity OBJECT-TYPE
    SYNTAX      OCTET STRING
    MAX-ACCESS  accessible-for-notify
    STATUS      current
    DESCRIPTION
```



```
        "The value of the community string field of an SNMPv1
         message containing a Trap PDU which is forwarded by a
         a proxy forwarder application using an SNMP version
         other than SNMPv1.  The value of this object SHOULD
         contain the value of the community string field from
         the original SNMPv1 message containing a Trap PDU as
         generated by an SNMPv1 agent."
    ::= { snmpCommunityMIBObjects 4 }

-- Conformance Information ******************************************

snmpCommunityMIBCompliances OBJECT IDENTIFIER
                            ::= { snmpCommunityMIBConformance 1 }
snmpCommunityMIBGroups      OBJECT IDENTIFIER
                            ::= { snmpCommunityMIBConformance 2 }

-- Compliance statements

snmpCommunityMIBCompliance MODULE-COMPLIANCE
    STATUS       current
```

```
    DESCRIPTION
        "The compliance statement for SNMP engines which
         implement the SNMP-COMMUNITY-MIB."

    MODULE       -- this module
        MANDATORY-GROUPS { snmpCommunityGroup }

        OBJECT           snmpCommunityName
        MIN-ACCESS       read-only
        DESCRIPTION     "Write access is not required."

        OBJECT           snmpCommunitySecurityName
        MIN-ACCESS       read-only
        DESCRIPTION     "Write access is not required."

        OBJECT           snmpCommunityContextEngineID
        MIN-ACCESS       read-only
        DESCRIPTION     "Write access is not required."

        OBJECT           snmpCommunityContextName
        MIN-ACCESS       read-only
        DESCRIPTION     "Write access is not required."

        OBJECT           snmpCommunityTransportTag
        MIN-ACCESS       read-only
        DESCRIPTION     "Write access is not required."

        OBJECT           snmpCommunityStorageType
```



```
        MIN-ACCESS       read-only
        DESCRIPTION     "Write access is not required."

        OBJECT           snmpCommunityStatus
        MIN-ACCESS       read-only
        DESCRIPTION     "Write access is not required."

    ::= { snmpCommunityMIBCompliances 1 }

snmpProxyTrapForwardCompliance MODULE-COMPLIANCE
    STATUS       current
    DESCRIPTION
        "The compliance statement for SNMP engines which
         contain a proxy forwarding application which is
         capable of forwarding SNMPv1 traps using SNMPv2c
         or SNMPv3."
    MODULE       -- this module
        MANDATORY-GROUPS { snmpProxyTrapForwardGroup }
    ::= { snmpCommunityMIBCompliances 2 }

snmpCommunityGroup OBJECT-GROUP
    OBJECTS {
        snmpCommunityName,
        snmpCommunitySecurityName,
```

```
        snmpCommunityContextEngineID,
        snmpCommunityContextName,
        snmpCommunityTransportTag,
        snmpCommunityStorageType,
        snmpCommunityStatus,
        snmpTargetAddrTMask,
        snmpTargetAddrMMS
    }
    STATUS       current
    DESCRIPTION
        "A collection of objects providing for configuration
         of community strings for SNMPv1 (and SNMPv2c) usage."
    ::= { snmpCommunityMIBGroups 1 }

snmpProxyTrapForwardGroup OBJECT-GROUP
    OBJECTS {
        snmpTrapAddress,
        snmpTrapCommunity
    }
    STATUS       current
    DESCRIPTION
        "Objects which are used by proxy forwarding applications
         when translating traps between SNMP versions.  These are
         used to preserve SNMPv1-specific information when
```



```
         translating to SNMPv2c or SNMPv3."
    ::= { snmpCommunityMIBGroups 3 }

END
```

6. 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.

7. Acknowledgments

This document is the result of the efforts of the SNMPv3 Working Group. The design of the SNMP-COMMUNITY-MIB incorporates work done by the authors of SNMPv2*:

Jeff Case (SNMP Research, Inc.)
David Harrington (Cabletron Systems Inc.)
David Levi (SNMP Research, Inc.)
Brian O'Keefe (Hewlett Packard)
Jon Saperia (IronBridge Networks, Inc.)
Steve Waldbusser (International Network Services)

8. Security Considerations

Although SNMPv1 and SNMPv2 do not provide any security, allowing community names to be mapped into securityName/contextName provides the ability to use view-based access control to limit the access of unsecured SNMPv1 and SNMPv2 operations. In fact, it is important for network administrators to make use of this capability in order to avoid unauthorized access to MIB data that would otherwise be secure.

Further, the SNMP-COMMUNITY-MIB has the potential to expose community strings which provide access to more information than that which is available using the usual 'public' community string. For this reason, a security administrator may wish to limit accessibility to the SNMP-COMMUNITY-MIB, and in particular, to make it inaccessible when using the 'public' community string.

When a proxy implementation translates messages between SNMPv1 (or SNMPv2c) and SNMPv3, there may be a loss of security. For example, an SNMPv3 message received using authentication and privacy which is subsequently forwarded using SNMPv1 will lose the security benefits of using authentication and privacy. Careful configuration of proxies is required to address such situations. One approach to deal with such situations might be to use an encrypted tunnel.

9. References

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

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

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

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

[5] McCloghrie, K. and M. Rose, "A Convention for Describing SNMP-based Agents", RFC 1303, February 1992.

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

[7] 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.

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

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

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

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

[12] Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Management Information Base for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1907, January 1996.

[13] Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Coexistence between Version 1 and Version 2 of the Internet-standard Network Management Framework", RFC 1908, January 1996.

[14] Levi, D. and B. Wijnen, "Mapping SNMPv2 onto SNMPv1 within a bi-lingual SNMP agent", RFC 2089, January 1997.

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

[16] Harrington, D. and B. Wijnen, "An Architecture for Describing SNMP Management Frameworks", RFC 2571, May 1999.

[17] Case, J., Harrington, D. and B. Wijnen, "Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)", RFC 2572, May 1999.

[18] Levi, D., Meyer, P. and B. Stewart, "SNMP Applications", RFC 2573, May 1999.

[19] Blumenthal, U. and Wijnen, B., "The User-Based Security Model for Version 3 of the Simple Network Management Protocol (SNMP)", RFC 2574, May 1999.

[20] Wijnen, B., Presuhn, R. and K. McCloghrie, "View-based Access Control Model for the Simple Network Management Protocol (SNMP)", RFC 2575, May 1999.

10. Editor's Addresses

Rob Frye
CoSine Communications
1200 Bridge Parkway
Redwood City, CA 94065
U.S.A.

Phone: +1 703 725 1130
EMail: rfrye@cosinecom.com

David B. Levi
Nortel Networks
3505 Kesterwood Drive
Knoxville, TN 37918
U.S.A.

Phone: +1 423 686 0432
EMail: dlevi@nortelnetworks.com

Shawn A. Routhier
Integrated Systems Inc.
333 North Ave 4th Floor
Wakefield MA 01880
U.S.A.

Phone: + 1 781 245 0804
EMail: sar@epilogue.com

Bert Wijnen
Lucent Technologies
Schagen 33
3461 GL Linschoten

Netherlands

Phone: +31 348 407-775
EMail: wijnen@lucent.com

A. Changes From RFC1908

- Editorial changes to comply with current RFC requirements.
- Added/updated copyright statements.
- Added Intellectual Property section.
- Replaced old introduction with complete new introduction/overview.
- Added content for the Security Considerations Section.
- Updated References to current documents.
- Updated text to use current SNMP terminology.
- Added coexistence for/with SNMPv3.
- Added description for SNMPv1 and SNMPv2c Message Processing Models and SNMPv1 and SNMPv2c Community-based Security Models.
- Added snmpCommunityMIB so that SNMPv1 and SNMPv2 community strings can be mapped into the SNMP Version Independent paramaters which can then be used for access control using the standard SNMPv3 View-based Access Control Model and the snmpVacmMIB.
- Added two MIB objects such that when an SNMPv1 notification (trap) must be converted into an SNMPv2 notification we add those two objects in order to preserve information about the address and community of the originating SNMPv1 agent.
- Included (and extended) from RFC2089 the SNMPv2 to SNMPv1 mapping within a multi-lingual SNMP Engine.
- Use keywords from RFC 2119 to describe requirements for compliance.
- Changed/added some rules for converting a MIB module from SMIv1 to SMIv2.
- Extended and improved the description of Proxy Forwarder behaviour when multiple SNMP versions are involved.

Full Copyright Statement

Copyright (C) The Internet Society (2000). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Acknowledgement

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